Expert Systems with Applications 42 (2015) 3297–3305

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Two-dimensional residual-space-maximized packing

Yanchao Wang, Lujie Chen *

Singapore University of Technology and Design, 20 Dover Drive, Singapore

ARTICLE INFO

Article history:
Available online 20 December 2014

Keywords:
Two-dimensional packing
Heuristics
Residual space
Residual-space-maximized packing

ABSTRACT

Many approaches exist for solving two dimensional rectangle-packing problems. Some rely on multiple heuristic policies to detect suitable packing positions. Others resort to searching for a sound packing sequence from a great number of variations. This paper describes a heuristic algorithm with only a single policy: maximize the residual space during packing, which ensures that rectangles to be packed will fit into the space with maximum likelihood. An efficient implementation is proposed to realize the policy. Experimental results based on openly available datasets demonstrate that the proposed algorithm is comparable to most state-of-the-art algorithms in space efficiency while is significantly faster in data processing. In the case of large-scale problems tested, it is the best by both evaluation metrics.

1. Introduction

Cutting and Packing (C&P) is a class of optimization problems with a wide range of applications in resource management (Dyckhoff, 1990). Over years of research, several subdivisions of C&P problems have been studied scholarly and their solutions have been implemented in the industry. Although C&P is known to be NP-hard (Non-deterministic Polynomial-time hard) in general (Garey & Johnson, 1979), existing methods can find near-optimal solutions to various problems; nevertheless, new approaches and algorithms are still emerging in the research field. Some offer even better solutions in broader scenarios, while others reduce the time and space overhead in implementation.

This paper pertains to two-dimensional (2D) rectangle packing, focusing on bin packing – single bin size bin packing problem (SBSBPP) and strip packing – open dimension problem (ODP) in the typology of C&P (Wäscher, Haußner, & Schumann, 2007). The objective of the former is to minimize the number of fixed-size bins and that of the latter is to minimize the overall height of a fixed-width, infinitely long bin. Specifically, the problems under investigation belong to the RF (Rotated, Free cutting) subtype as classified by Lodi, Martello, and Vigo (1999), where input rectangles can be rotated and packing is not constrained by guillotine cutting.

A 2D rectangle-packing algorithm is developed based on a single policy: the residual space (RS), i.e. the unused region, should be maximized during each step of packing. It will be shown that an efficient implementation exist for such a residual-space-maximized packing (RSMP) policy. RSMP can produce comparable results to existing algorithms but runs considerably faster. The related work to this study is reviewed in Section 2. The principle of RSMP is described in Section 3. Comparisons with existing algorithms based on benchmark datasets are presented in Section 4. Conclusion and future work of the proposed approach are presented in Section 5.

2. Related work

One of the early approaches to 2D packing is the so-called exact method based on integer programming formulation (Beasley, 1985; Martello & Vigo, 1998). It attempts to find the packing positions of rectangles by solving a set of linear equations, constructed to optimize certain criteria. The number of equations to solve is related to the number of rectangles to pack; thereby the computational cost is relatively high.

A different approach, known as the heuristic approach, produces solutions with relatively less computational overhead. Heuristic algorithms are motivated by intuitive experience, e.g. rectangles are piled to the bottom-left (BL) corner of a bin by the BL heuristic (Baker, Coffman, & Rivest, 1980). Performance of an algorithm depends on the efficacy of the heuristic. Many heuristic algorithms consist of a placement method, used to place a rectangle at a specific location in a bin, and a sequence-generation strategy, to create various sequences by changing the order of the input rectangles. Bottom-left (BL) (Baker et al., 1980), bottom-left-fill (BLF) (Chazelle, 1983), improved bottom-left (IBL) (Liu & Teng, 1999) and difference process DP (Lai & Chan, 1997) are widely used

* Corresponding author.
E-mail addresses: wangyanchao1988@gmail.com (Y. Wang), chenlujie@sutd.edu.sg (L. Chen).

http://dx.doi.org/10.1016/j.eswa.2014.12.021
0957-4174/

placement methods. Methods such as floor-ceiling (FC) and touching perimeter (TP) have also been reported (Lodi et al., 1999).

Sequence generation is motivated by the fact that a placement method would produce different results if the order of the input rectangles is varied. When different sequences are tested, the best result can be treated as the solution. To date, lots of strategies have been proposed, such as simulated annealing (SA) (Dowsland, 1993) and genetic algorithm (GA) (Jakobs, 1996; Liu & Teng, 1999), artificial neural network (Dagli & Poshyanonda, 1997), tabu search (TS) (Lodi et al., 1999), unified tabu search (UTS) (Bennell, Lee, & Potts, 2013; Lodi, Martello, & Vigo, 2004), greedy randomized adaptive search procedure (GRASP) (Alvarez-Valdés, Parreño, & Tamarit, 2005), sequential value correction (SVC) (Belov, Scheithauer, & Mukhacheva, 2008), iterative maximal area (IMA) (Hayek, Moukrim, & Negre, 2008), iterative doubling binary search (IDBS) (Wei, Oon, Zhu, & Lim, 2011), single- and multi-crossover genetic algorithm (SGA and MXGA) (Bennell et al., 2013). Some of these algorithms have achieved near-optimal solutions in certain scenarios; however, the drawback is the prolonged computation time due to many trials of different sequences.

There are alternatives to sequence generation relatively independent from a placement method. For example, Burke, Kendall, and Whitwell (2004) proposed a best-fit (BF) heuristic algorithm that could actively select a suitable sized rectangle to pack, which makes it a placement method that generates a sequence during packing. BF can also be combined with a sequence-generation strategy to improve the performance (Burke, Kendall, & Whitwell, 2006). Variations of BF have been proposed, such as the bidirectional best-fit (BBF) (Aşık & Özcan, 2009) and modified bidirectional best-fit (BBFM) (Özcan, Kai, & Drake, 2013) algorithms. Among these, BBFM is the best in terms of space-efficiency, owing to a large number of (6912) policy combinations incorporated. The supposedly most suitable policy is applied at each phase of packing but the computational cost is heavy due to multiple levels of nested comparison.

Some recent publications suggested that effective algorithms could be based on only a few policies. The fast heuristic (FH) algorithm (Leung & Zhang, 2011) applied a scoring policy and the binary search heuristic algorithm (BSHA) (Zhang, Wei, Leung, & Chen, 2013) incorporated a primary policy that ensures the smoothness of the unused region's envelop. Better results than most existing algorithms were achieved and the processing time was shorter in some scenarios than the algorithms of many policies (Aşık & Özcan, 2009; Özcan et al., 2013). However, the processing time was not proportional to the number of rectangles: some small-scale tasks might take longer to process than large-scale problems.

3. Principle

The heuristic policy of the proposed residual-space-maximized packing (RSMP) algorithm is motivated by an observation: the most problematic issue of packing is where to put big rectangles. If not properly handled, they may occupy a great number of bins (in bin packing) or reach considerable piled height (in strip packing) with many small spaces unused. Hence, it is reasonable to hypothesize that the best packing position of a rectangle is the one that maximizes the residual space (RS). In this paper, a RS is defined in the same way as in Lai and Chan (1997): it is the largest rectangular area that can be obtained in a free area, where the two areas have at least one mutual edge. In so doing, the chance that subsequent rectangles can fit into the space is maximized. The hypothesis also suggests that it is reasonable to sort the input rectangles in an order of big to small (Hopper & Turton, 2001): pack the big ones before the small ones.

Guided by the hypothesis, RSMP first orients each input rectangle so that its height is not longer than its width; then creates three sequences: height descending (break a tie by descending width), width descending (break a tie by descending height) and area descending (a tie is left as is). The sequences are tested for packing and the best result is chosen as the solution. During the processing of each rectangle, every possible packing position is tested to generate a series of RSs. The position that results in the largest RS is chosen as the final decision. If there is a tie, i.e. two positions producing the same largest RS, the final decision will be based on the comparison of the second largest RS; so on and so forth. This comparison strategy is the RSMP policy.

Superficially, the policy of RSMP seems to have little difference from packing a rectangle in the smallest suitable space (Gonçalves, 2007); nevertheless, the example shown in Fig. 1 illustrates that the two strategies produce quite different results. As can be seen, RSs may overlap; thereby a rectangle packed in the smallest space may occupy part of a larger space and break it into smaller pieces. RSMP examines all possible packing positions and takes the one that ensures the largest RS.

Computationally, RSMP has to keep track of all RSs and to perform multiple tests to find the packing position of each rectangle. The time and space overhead must be carefully managed in order to achieve an efficient algorithm. The proposed implementation consists of the following steps for bin packing. Few modifications are needed for strip packing as will be described later.

1	Preprocessing: create three input sequences. For each sequence, run RSMP as follows.
2	Initialize an area-descending RS list with one element: a RS of the bin size.
3	For each rectangle, find the best packing position. A rectangle larger than the bin is discarded.
3.1	Find the smallest suitable RS to pack the rectangle. If not found, a new RS equal to an empty bin is added to the front of the list.
3.2	Test all suitable RSs, from the smallest to the largest. For each RS, try packing the rectangle in eight configurations (4 corners $\times$ 2 orientations). Keep track of the best packing position based on the RSMP policy.
3.3	Update the list based on the best packing position.

3.1. Difference process (DP)

RSMP applies DP (Lai & Chan, 1997) to generate RSs from a free space partly occupied by a rectangle. Fig. 2 illustrates this fundamental process, repeatedly used in step 3. A RS is rectangular, with

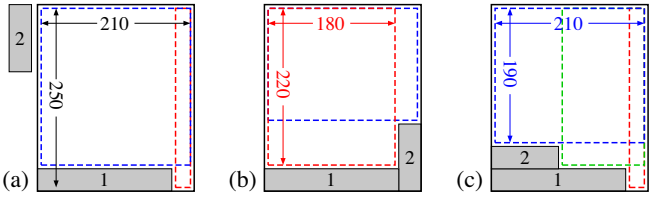

Fig. 1. (a) Two rectangles (the first 180 by 30 and the second 90 by 30) are to be packed in a bin of 210 by 250. The blue and red regions indicate two RSs after the first rectangle is packed. (b) Based on the smallest-suitable-space policy (Gonçalves, 2007), the second rectangle will be packed vertically at the bottom-right corner. The subsequent largest RS is 180 by 220. (c) RSMP will pack the second one horizontally because the largest RS in this position is 210 by 190, larger than that obtained in (b). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

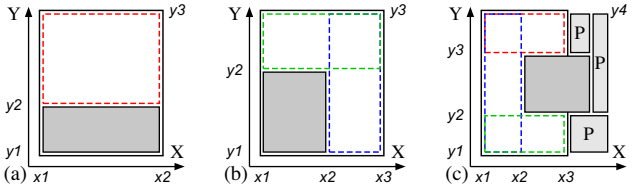

Fig. 2. Examples of RSs produced by different relative positions of a free space (dark solid frame) and a rectangle (gray). (a) A rectangle occupying one side of the free space generates one RS. (b) A rectangle packed at a corner of the free space generates two RSs. (c) A rectangle partly overlapping one side of the free space generates three RSs. This situation may occur due to positions of the packed rectangles, labeled by P.

coordinates determined by those of the free space and the rectangle. Depending on the situation, one, two or three RSs may be produced and they can overlap each other.

3.2. Find the best packing position

Step 3 is the key to RSMP. The RS list is maintained in area-descending order throughout the process.

Step 3.1: the smallest suitable RS in the list for an input rectangle can be located quickly. First, a binary search for the rectangle's area in the list gives a RS with an equal or larger area. Then, from this RS to the front of the list (from small to big RSs), locate the smallest one that can accommodate the rectangle.

Step 3.2: from the located RS to the front of the list, each RS is tested to pack the rectangle in eight configurations: the original and rotated rectangle in four corners, as shown in Fig. 3. The test of each configuration involves finding all RSs affected (partly occupied) by the rectangle, calculating the generated new RSs by DP, and creating a temporary list merging the new ones with those in the RS list unaffected by the rectangle. The temporary list is compared with that produced by the current best packing position, which is initialized by the first feasible packing position. If a test result is better based on the RSMP policy, the current best is replaced.

The computational cost for finding the best packing position of a rectangle would be considerable if the above procedure was implemented as described. In practice, several conditions may be exploited to reduce the cost. First, if a rectangle is a square or it can fit into a RS in one orientation (original or rotated) only, half of the tests are needed. Second, if one dimension of a rectangle and a RS is the same, the number of corners to test is reduced by half in the corresponding orientation; if both dimensions are the same, one test suffices.

Third and the most important, a data structure of the current best result should support two-level comparison. Table 1 shows a data structure used in this study. RS_p and Conf. record the best packing position. RS_l and Arr_a are for two-level comparison with a test result. RS_l, which may or may not be the same as RS_p, is the largest RS in the list affected by the rectangle. Arr_a stores, in

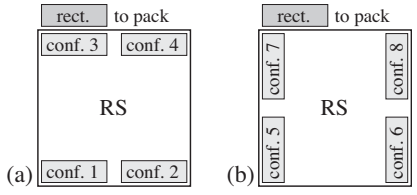

Fig. 3. Eight packing configurations (conf. 1–8) of a rectangle in a RS. The (a) original and (b) rotated rectangle can be packed at four corners of the RS.

Table 1
Data structure of the current best packing position of a rectangle.

RS_p	RS to pack a rectangle
Conf.	Packing configuration (conf. 1–8)
RS_l	The largest affected RS in the RS list
Arr_a	An array of descending RS area

descending order, the area of unaffected RSs smaller than RS_l and those of the generated RSs from the affected ones.

It is time-consuming to calculate Arr_a; therefore the first-level comparison is based on RS_l, which can be obtained with little computational overhead: search from the front of the RS list for one that is affected by the rectangle. A difference in RS_l would produce a winner of the comparison and calculation of Arr_a is saved if the current best wins (an example shown in Fig. 4). Only when there is a tie, the second-level comparison based on Arr_a is required.

Last but not the least, by default, RSs from the smallest suitable one to the front of the list shall be tested for packing but as the current best is updated, the largest RS to test should be set to RS_l of the current best because packing in a RS larger than RS_l would definitely generate a packing worse than the current best considering the RSMP policy.

The above procedure significantly optimizes the search for the best packing position of each rectangle.

3.3. Update the RS list

Based on the best packing position, the affected RSs are removed from the RS list and then generated new RSs are insert into it. Prior to insertion, each new RS must pass three checks. First, it must be able to accommodate the smallest input rectangle. Second, it is not contained in any other new RS. Third, it is not contained in any RS in the list. New RSs that fail on any of the check can be discarded.

After the list is updated, the next rectangle in the input sequence is fed into step 3 for processing. The algorithm completes one sequence when all rectangles are packed. The three sequences are run in the order of height, width and area descending; the best result is saved as the solution. In some problems, the optimal solution is prior known. If so and the result of a sequence is equivalent to the optimal solution, subsequent sequences are not run.

3.4. An example

Based on the example of Fig. 1(c), this section shows the calculation of the best packing position of a third rectangle (90 by 30). The smallest suitable RS is the red one shown in Fig. 5(a). Only two configurations are feasible, as indicated in Fig. 5(b) and (c). They are conf. 5 and 7 according to the convention defined in Fig. 3: the height of the original rectangle is shorter than the width.

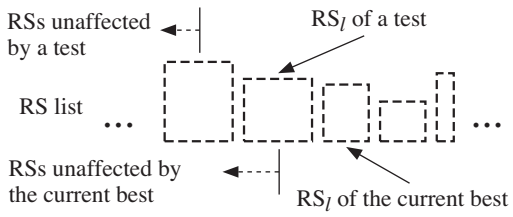

Fig. 4. The current best packing position affects a RS_l smaller than that affected by a test packing position. As a result, the maximal different RS is the red one on the left. It is unaffected by the current best packing position but partly occupied by the test; hence, the current best is definitely the winner of the comparison and calculation of Arr_a of the test is unnecessary.

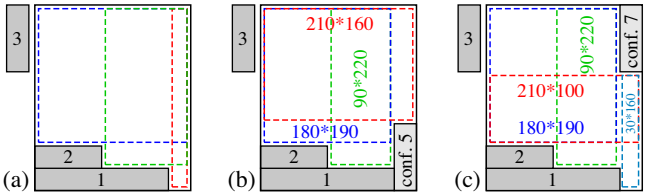

Fig. 5. Test of the smallest suitable RS, the red one in (a). Resultant RSs based on (b) conf. 5 and (c) conf. 7. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Conf. 5 is the first feasible packing position and is assigned to the current best with Arr_a calculated. Conf. 7 produces the same largest RS (180∗190) but a smaller second largest RS; therefore, the current best remains to be conf. 5.

The next RS to test is the green one shown in Fig. 5(a). Conf. 1 (Fig. 6(a)) produces a larger maximal RS than conf. 5 of Fig. 5(b). The current best is updated accordingly. In fact, in this configuration, the largest RS (blue) is not affected; hence, RS_l of the current best is the green one in Fig. 5(a). Conf. 2 (Fig. 6(b)) does not affect the largest RS either but produces a smaller second largest RS. The remaining configurations (Fig. 6(c)) affect the blue RS, thereby fail in the first-level comparison with the current best, while Arr_a is not calculated.

The largest RS (blue) in Fig. 5(a) will not be tested for packing because RS_l of the current best (the green RS) is smaller. Any configuration in the blue RS would fail to compete with the current best. Consequently, the best packing position of the third rectangle is shown in Fig. 6(a).

3.5. Strip packing

In strip packing, the bin width is fixed while the bin height is flexible. To adapt the RSMP algorithm described in the previous section to strip packing, one may set the bin height to a large value, enough to accommodate all rectangles. The value serves as a virtual bound. In the test of a RS to pack a rectangle, if the top side of the RS is the virtual bound, the top corners are not tested. The procedure stays the same as in bin packing.

However, a solution thus obtained is not necessarily the most space efficient that RSMP can achieve due to the large space beneath the virtual bound. As that space is the largest RS, rectangles tend to be packed horizontally to maximize it. This prevents vertical packings that may be more space efficient occasionally. To deal with the issue, an additional procedure is applied, in which different bin heights are systemically tested, as illustrated in Fig. 7. (A similar procedure was described by Zhang et al. (2013).) Assuming that H_1 is the piled height obtained using the virtual bound as the bin height, the next bin height to test is H_2, halfway between H_1 and a theoretical bound (sum of all rectangles' area divided by the bin width). From this stage onwards, the virtual bound is no longer

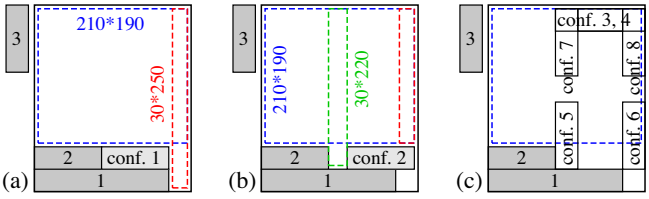

Fig. 6. Test of the green RS in Fig. 5(a). Resultant RSs based on (a) conf. 1 and (b) conf. 2. The largest RS (blue) is not affected. (c) Confs. 3–8 affect the largest RS. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

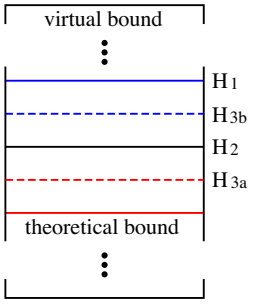

Fig. 7. A bisection strategy for finding the smallest feasible height of strip packing.

applicable and all four corners are tried in a RS test. If the input rectangles can be packed in one bin of height H_2 based on any of the three sequences, H_2 is a better solution than H_1; then RSMP is run again using H_{3a} as the bin height, halfway between H_2 and the theoretical bound. On the contrary, if the rectangles cannot be packed in one bin, H_{3b} is set as the bin height, halfway between H_1 and H_2. The process continues until to a predetermined resolution and the smallest height that is able to accommodate all rectangles is the final solution of the strip packing.

4. Results and discussions

The proposed RSMP algorithm has been tested on various randomly generated and widely used benchmark datasets. The randomly generated datasets contribute to the understanding of the relative performance of RSMP with respect to the three input sequences (Section 4.1). The benchmark datasets are used to compare RSMP with existing algorithms in terms of bin packing (Section 4.2), strip packing (Section 4.3), and large-scale problems (Section 4.4). Results of the existing algorithms are obtained from the published papers and are valid for direct comparison: 2D, non-guillotine, rotation-allowed, rectangle-packing with no other constraint. Analysis of computational cost of RSMP is given in Section 4.5. Our implementation of RSMP is realized in C++ and the results were generated on a HP Pavilion desktop (Intel Core i7, 3.4 GHz CPU and 8 GB RAM).

4.1. Performance with respect to sequence

Randomly generated rectangles are packed by RSMP based on the height, width and area-descending sequences. A typical dataset is created using the following parameters while other datasets show similar results. In bin packing, the bin width, W, is fixed to 100; the bin height, H, is varied from 100 to 1000. The number of rectangles, n, increases with H, from 100 to 1000. In strip packing, the bin width is fixed to 100 and n varies from 50 to 500. Among the input rectangles to bin packing, 80% have their width in $[2W/3, W]$ and height in $[1, W/2]$ (group 1), 10% have width in $[W/2, W]$ and height in $[W/2, W]$ (group 2), and the rest 10% have width in $[1, W/2]$ and height in $[1, W/2]$ (group 3) randomly generated. They compose a diverse collection of long bars, big and small near-square rectangles. In striping packing, the ratios of groups 1 and 3 are swapped.

Fig. 8 shows the average results over 5000 instances. In bin packing (Fig. 8(a)), all three sequences are able to produce a good proportion of the best solution because the evaluation metric is rounded off to an integer number of bins consumed. In strip packing (Fig. 8(b)), the height-descending sequence outperforms the other two in producing a lower piled height. Based on visual inspection of the packing patterns, we believe that the advantage is due to the relatively uniform size of consecutive rectangles in

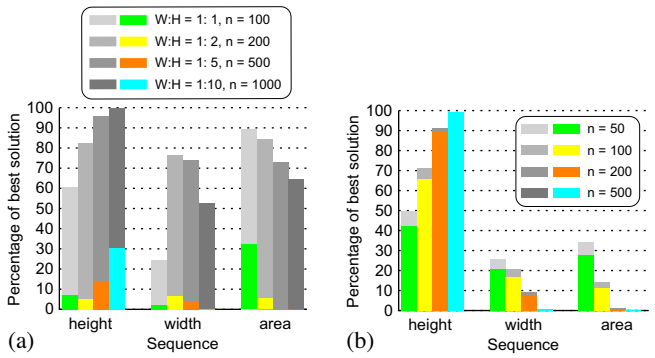

Fig. 8. (a) Bin-packing and (b) strip-packing results over 5000 instances. The gray and color bars indicate respectively the best and the unique best solution of the three sequence trials. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the height-descending sequence. As mentioned in Section 3, a pre-processed rectangle has a shorter height than width; therefore, height descending roughly ensures area and width descending. In contrast, the area-descending sequence has weaker constraint on the height and width; rectangles of quite different aspect ratio may be ordered closely. Consequently, more tiny unusable RSs are generated. The width-descending sequence bears similar disadvantage as it only weakly constrains the area and height.

Despite the fact that the height-descending sequence is the best on average, in some situations, especially when n is small, the other two sequences can produce better solutions, as indicated by the color bars in Fig. 8. For this reason, RSMP incorporates all three sequences to find the best solution.

4.2. Bin packing results

Two groups of bin packing algorithms are compared. In the first group, four algorithms (TS, UTS, SGA and MXGA) are compared with RSMP based on bin packing datasets given in Berkey and Wang (1987), Martello and Vigo (1998). The optimal solution of each instance is unknown. Results of TS are obtained from Lodi et al. (1999) and those of UTS, SGA and MXGA are from Bennell et al. (2013). The version of UTS compared is UTS_{BFB} proposed by Bennell et al. (2013). Table 2 shows the performance of the algorithms on each data class, consisting of several instances. TS and UTS produce one and three best solutions among the ten classes respectively. SGA does not produce any best solution. MXGA and RSMP have achieved five best solutions each.

In terms of the processing time on each instance, TS was set to run for 60 s (Lodi et al., 1999) (implemented in Fortran 77, test machine: Silicon Graphics INDY R10000sc, 195 MHz CPU); UTS, SGA and MXGA were set to run for 120 s (Bennell et al., 2013) (implemented in C++, test machine: Pentium 4, 2 GHz CPU, 2 GB RAM); RSMP only needed about 1 ms (implemented in C++, test machine: Intel Core i7, 3.4 GHz CPU, 8 GB RAM). Despite the fact that RSMP was executed on a faster computer, the drastic difference in computational speed suggests an advantage of RSMP in efficiency.

Table 2
Bin packing results of TS, UTS, SGA, MXGA and RSMP. A number indicates the average ratio of the number of packed bins and the lower bound (Dell'Amico et al., 2002), the smaller the number, the better the performance. A bold number indicates the best solution of each class.

Class	TS	UTS	SGA	MXGA	RSMP
I	1.044	1.039	1.039	**1.034**	1.046
II	1.026	**1.020**	1.073	1.027	**1.020**
III	1.074	1.075	1.087	1.068	**1.066**
IV	1.040	**1.033**	1.053	1.047	**1.033**
V	1.062	1.079	1.073	**1.059**	1.062
VI	1.104	1.083	1.110	1.093	**1.067**
VII	1.086	1.093	1.103	**1.080**	1.113
VIII	1.088	1.094	1.097	**1.079**	1.105
IX	1.008	**1.007**	1.008	**1.007**	**1.007**
X	**1.068**	1.072	1.080	1.072	1.079

In the second group, three algorithms (HBP (Boschetti & Mingozzi, 2003), ATS-BP (Harwig, Barnes, & Moore, 2006) and IMA (Hayek et al., 2008)) are compared with RSMP based on bin packing datasets given in Berkey and Wang (1987), Martello and Vigo (1998). Each class of the dataset has five groups and each group has 10 instances. Results shown in Table 3 suggests that RSMP consumes slightly more bins on average than the other algorithms. This is largely due to the fact that the number of rectangles is small; so by testing many sequences HBP, ATS-BP and IMA produce better results than RSMP, which tests three sequences only. In terms of the average processing time, HBP took 1824.2 ms (Intel Pentium 3, 933 MHz CPU), ATS-BP took 70210 ms (CPU specification unavailable), IMA took 318.6 ms (Intel Pentium 4, 2.66 GHz CPU), and RSMP took 2.5 ms (Intel Core i7, 3.4 GHz CPU).

4.3. Strip packing results

Several algorithms are compared with RSMP based on strip packing datasets given in Hopper and Turton (2001), Burke et al. (2004). The optimal solution of each instance is known. Results of GA + BLF and SA + BLF are obtained from Burke et al. (2004), those of BF + SA, BF + GA, BF + TS are from Burke et al. (2006), those of IDBS are from Wei et al. (2011), those of BBF and BBFM are from Özcan et al. (2013), and those of BSHA are from Zhang et al. (2013).

Table 4 shows the results of the algorithms and Fig. 9 is a summary of the performance. The metric in Fig. 9 is the average extra height over all instances; while extra height is the difference between the piled height and the optimal solution. Excluding the optimal solution values in the plot highlights the difference among the algorithms. RSMP outperforms GA + BLF and SA + BLF, is comparable to BF + SA, BF + GA, BF + TS and BBF, and is not as good as IDBS, BBFM and BSHA. BBFM (Özcan et al., 2013) incorporates a large number of (6912) policy combinations while BSHA (Zhang et al., 2013) introduces randomness in sequences. Their performance comes with a noticeable overhead in running time, as shown in Table 5. IDBS reached optimal solution in all instances according to Wei et al. (2011); it was set to run for 100 s; on average the optimal solution appeared at 310 ms after start. (It is not included in Table 5 as no published data are available for direct comparison.) Although it is difficult to ascertain to what extent different hardware platforms (Table 5) influence the running time, RSMP appears to be more time-efficient than BBFM and BSHA. RSMP is about twice faster than BBF with slightly better performance (Fig. 9). The other algorithms (GA + BLF, SA + BLF, BF + SA, BF + GA and BF + TS) were tested on much older machines and the published results are not directly comparable in time efficiency.

4.4. Large-scale problems

The efficacy and efficiency of RSMP are best demonstrated in large-scale problems. BL by Zhang et al. (2013), BBF (Aşık & Özcan, 2009), BSHA (Zhang et al., 2013) and RSMP are compared based on the large-scale datasets created by Pinto and Oliveira (2005). The optimal solution of each instance is known. As shown in Table 6, RSMP produces the best results except for the first instance of 50 rectangles; it has achieved the optimal solution in all instances for n larger than 100.

Table 3
Bin packing results of HBP, ATS-BP, IMA and RSMP. A number indicates the average number of packed bins used in each group (10 instances). The smaller is the number, the better is the performance.

Class	n	HBP	ATS-BP	IMA	RSMP	Class	n	HBP	ATS-BP	IMA	RSMP
1	20	6.6	6.6	6.6	6.7	6	20	1.0	1.0	1.0	1.0
	40	12.9	12.9	12.9	13.1		40	1.7	1.6	1.7	1.8
	60	19.5	19.5	19.5	19.6		60	2.1	2.1	2.1	2.1
	80	27.0	27.0	27.0	27.1		80	3.0	3.0	3.0	3.0
	100	31.3	31.4	31.3	31.4		100	3.4	3.4	3.2	3.3
	Avg.	19.46	19.48	19.46	19.58		Avg.	2.24	2.22	2.20	2.24
2	20	1.0	1.0	1.0	1.0	7	20	5.2	5.2	5.2	5.4
	40	1.9	1.9	1.9	2.0		40	10.5	10.4	10.4	10.7
	60	2.5	2.5	2.5	2.5		60	15.1	14.6	14.7	15.3
	80	3.1	3.1	3.1	3.1		80	21.8	21.3	21.2	21.9
	100	3.9	3.9	3.9	3.9		100	25.9	25.5	25.3	26.2
	Avg.	2.48	2.48	2.48	2.50		Avg.	15.70	15.40	15.36	15.90
3	20	4.7	4.7	4.7	4.7	8	20	5.3	5.3	5.3	5.4
	40	9.4	9.4	9.4	9.5		40	10.5	10.4	10.4	10.7
	60	13.5	13.6	13.5	13.6		60	15.4	15.1	15.0	15.5
	80	18.4	18.6	18.4	18.6		80	21.3	20.8	20.8	21.4
	100	22.2	22.3	22.2	22.2		100	26.3	26.0	25.7	26.4
	Avg.	13.64	13.72	13.64	13.72		Avg.	15.76	15.52	15.44	15.88
4	20	1.0	1.0	1.0	1.0	9	20	14.3	14.3	14.3	14.3
	40	1.9	1.9	1.9	1.9		40	27.5	27.6	27.5	27.5
	60	2.5	2.4	2.5	2.5		60	43.5	43.5	43.5	43.5
	80	3.2	3.2	3.1	3.2		80	57.3	57.3	57.3	57.3
	100	3.8	3.8	3.7	3.7		100	69.3	69.3	69.3	69.3
	Avg.	2.48	2.46	2.44	2.46		Avg.	42.38	42.40	42.38	42.38
5	20	5.9	5.9	5.9	5.9	10	20	4.1	4.1	4.1	4.2
	40	11.5	11.4	11.4	11.5		40	7.3	7.3	7.3	7.3
	60	17.5	17.5	17.4	17.5		60	10.0	9.9	10.1	10.0
	80	24.0	23.9	23.9	24.0		80	12.8	12.8	12.8	13.0
	100	28.0	28.0	27.9	28.3		100	16.0	15.9	15.8	16.1
	Avg.	17.38	17.34	17.30	17.44		Avg.	10.04	10.00	10.02	10.12

Table 4
Strip packing results of several algorithms. The numbers under each algorithm are the piled height and those under Opt. H are the theoretical optimal solution. "–" indicates that packing cannot be finished in 10^6 s. IDBS reached optimal solution in all instances (Wei et al., 2011); for simplicity, it is not included in the table.

Inst.	*n*	Opt. H	GA + BLF	SA + BLF	BF + SA	BF + GA	BF + TS	BBF	BBFM	BSHA	RSMP
C1P1	16	20	20	20	20	20	20	21	20	20	21
C1P2	17	20	21	21	20	21	21	21	21	20	21
C1P3	16	20	20	20	20	20	20	21	21	20	21
C2P1	25	15	16	16	16	16	16	16	16	15	16
C2P2	25	15	16	16	16	16	16	17	15	15	16
C2P3	25	15	16	16	16	16	16	16	16	15	16
C3P1	28	30	32	32	31	31	31	32	30	30	31
C3P2	29	30	32	32	31	32	32	33	31	30	31
C3P3	28	30	32	32	31	31	31	33	32	30	31
C4P1	49	60	64	64	61	62	62	62	62	60	61
C4P2	49	60	63	64	61	62	62	63	61	60	61
C4P3	49	60	62	63	61	62	61	62	61	60	61
C5P1	73	90	95	94	91	92	92	91	91	90	92
C5P2	73	90	95	95	91	92	92	92	91	90	91
C5P3	73	90	95	95	92	92	92	92	91	90	92
C6P1	97	120	127	127	122	122	122	123	121	120	122
C6P2	97	120	126	126	121	121	121	123	122	120	121
C6P3	97	120	126	126	122	122	122	123	121	120	122
C7P1	196	240	255	255	244	245	245	243	242	240	243
C7P2	197	240	251	253	244	244	244	242	242	240	243
C7P3	196	240	254	255	245	245	245	243	241	240	242
N1	10	40	40	40	40	40	40	40	40	40	40
N2	20	50	51	52	50	50	50	52	50	50	53
N3	30	50	52	52	51	52	51	52	52	50	52
N4	40	80	83	83	82	83	83	82	82	80	83
N5	50	100	106	106	103	104	103	103	102	100	106
N6	60	100	103	103	102	102	102	102	101	100	102
N7	70	100	106	106	104	104	105	106	105	101	102
N8	80	80	85	85	82	82	82	82	81	80	82
N9	100	150	155	155	152	152	152	152	151	150	154
N10	200	150	154	154	152	152	152	151	151	150	152
N11	300	150	155	155	153	153	153	151	150	150	152
N12	500	300	313	312	306	306	306	302	302	300	305
N13	3152	960	–	–	964	964	964	964	960	960	962

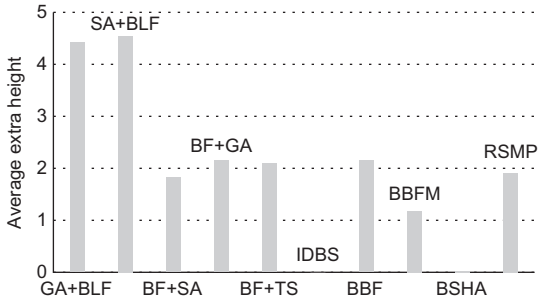

Fig. 9. Summary of the performance of the algorithms in Table 4 and that of IDBS. The average extra height of IDBS is 0 and that of BSHA is 0.03.

Table 5
Processing time of BBF, BBFM, BSHA and RSMP in millisecond spent on datasets C1–C7 in Table 4.

Class	n	BBF	BBFM	BSHA	RSMP
C1	16, 17	3	178	23	1
C2	25	6	381	93	2
C3	28, 29	7	523	1163	5
C4	49	17	2248	2583	6
C5	73	29	4858	4157	9
C6	97	45	8707	12683	14
C7	196, 197	126	38713	101613	66
Language		–		C++	C++
CPU		Intel i7		–	Intel i7
CPU (GHz)		3.2		2.6	3.4
RAM (GB)		16		0.5	8

Table 6
Results of large-scale strip packing problems. The optimal solution is known to be 600 for all instances. The bold numbers indicate the best solutions achieved among the four algorithms.

n	BL	BBF	BSHA	RSMP
50	674	643	**601**	612
100	679	645	610	**608**
500	692	618	**600**	**600**
1000	690	**600**	**600**	**600**
5000	687	**600**	**600**	**600**
10,000	681	**600**	**600**	**600**
15,000	660	**600**	**600**	**600**

Table 7
Processing time of BBF, BSHA and RSMP in millisecond tested on the large-scale datasets.

n	BBF	BSHA	RSMP
50	250	112480	10
100	610	1329120	37
500	6230	12680	12
1000	17870	330	21
5000	413940	3520	87
10,000	1780910	12110	234
15,000	4237500	26510	226
Language	Java	C++	C++
CPU	Intel Core 2	–	Intel i7
CPU (GHz)	1.86	2.6	3.4
RAM (GB)	2	0.5	8

Table 7 shows the algorithms' processing time for each instance. (The time of BL was not provided in Zhang et al. (2013).) The time of BBF increases with n but that of BSHA does not. BSHA may spend a long time even on a small-scale problem. The time of RSMP is in general related to n but for the instance $n = 100$, RSMP spent 37 ms, longer than that for $n = 500$ and 1000. We found that when $n = 100$ the strip packing procedure (Section 3.5) ran several times before settling on the final solution, whereas for $n = 500$ and 1000, the optimal solution was found in the initial run; hence, less time was consumed. Overall, RSMP is clearly faster than BBF and BSHA.

Based on the comparisons, one can see that RSMP is able to produce space-efficient solutions to various packing problems and is faster than many advanced algorithms.

4.5. Computational cost

The most time-consuming step in RSMP is the second-level comparison of RS, which requires the calculation of Arr_a (Section 3.2). We compare the proposed efficient implementation with one that searches for the best packing position of a rectangle by brute force, i.e. all feasible RSs are tried, all eight configurations of each RS are tested, and Arr_a is always calculated by DP. The results are shown in Fig. 10, which is the worst case of 100 randomly generated input rectangle sequences. The worst-case metric is the number of RS generated in total. While the results are based on a particular strip packing dataset, other datasets show similar trend.

The number of RS generated (red line) is the same for both implementations, suggesting identical results. Toward the end of a packing sequence, be it 200 or 500 input rectangles, the number of RS stops increasing. This is due to the fact that small rectangles are packed at the end of a sequence and they generate even smaller RSs. A RS that cannot accommodate the smallest rectangle is discarded; hence, the number of RS tends to drop.

The number of DP required (blue line) is a direct indication of the computational cost. As can be seen in Fig. 10(a) and (c), the number of DP in the brute force implementation increases with the number of RS. Toward the end of the packing sequence, the number of DP increases dramatically because small rectangles can be packed in a great number of RSs and many tests are performed. In contrast, the number of DP in the efficient implementation does not increase with the number of RS (Fig. 10(b) and (d)), owing to the fact that redundant calculation of Arr_a is avoided.

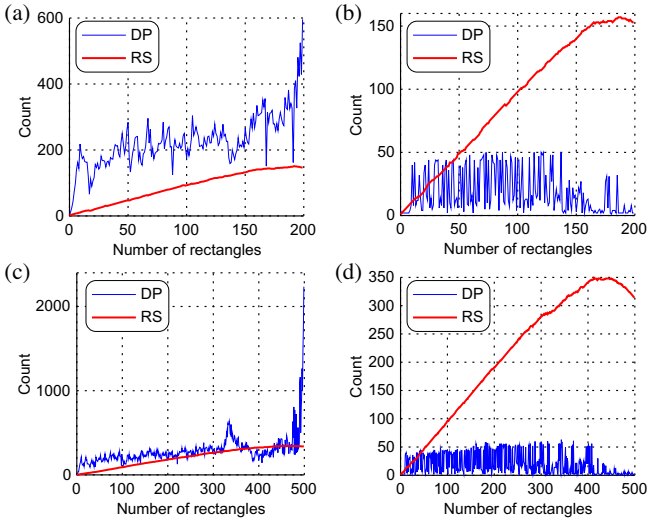

Fig. 10. The number of DP required to determine the best packing position of the nth rectangle and the number of RS generated after packing the nth rectangle. One DP is the partition of one RS by a rectangle, as illustrated in Fig. 2. (a) 200 input rectangles, packed by the brute force implementation. (b) 200 input rectangles, packed by the efficient implementation. (c) 500 input rectangles, packed by the brute force implementation. (d) 500 input rectangles, packed by the efficient implementation, in which the number of DP required for each rectangle does not exceed 63.

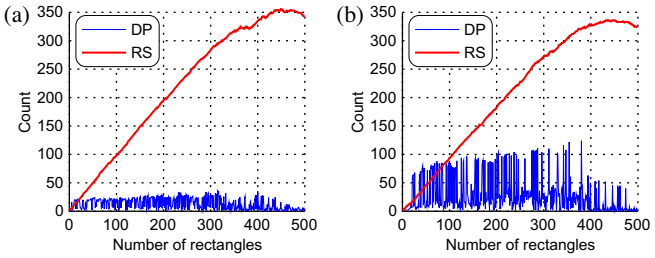

Fig. 11. Results based on the same dataset as that used to produce Fig. 10(d), except that the width of the bin in (a) is halved and that in (b) is doubled. The number of DP required for each rectangle does not exceed 36 in (a) and 124 in (b).

There is an upper bound of the number of DP required for each rectangle in the efficient implementation. The upper bound increases with the width of a strip bin, as shown in Fig. 11. A small width produces a low upper bound (Fig. 11(a)) because in a narrow bin the probability that a rectangle overlaps many RSs is relative small. In contrast, such probability is higher in a wider bin of a larger width (Fig. 11(b)). What is important is that the upper bound exists and it is not related to the number of RS.

In a worst-case scenario, each rectangle affects several RSs as they can overlap and each affected RS may be partitioned into multiple RSs; therefore, the number of RSs can reach n^2, where n is the number of input rectangles. The time complexity for processing one rectangle is $\mathcal{O}(n^4)$ because a rectangle may be tested in every RS [$\mathcal{O}(n^2)$] and each test would involve DP with all RSs [$\mathcal{O}(n^2)$]. The overall time complexity is $\mathcal{O}(n^5)$. In an average-case scenario, each rectangle affects a limited number of RSs; the number of RSs is linear to n as shown by the previous experiment. In addition, there are two situations due to the upper bound discussed in the previous paragraph. Let c be the upper bound of the number of DP needed to find the best packing position of each rectangle. If time spent on c is larger than that spent in finding the smallest RS to pack a rectangle [binary search $\mathcal{O}(\log n)$], the average-case time complexity is $\mathcal{O}(n)$ because processing pertaining to each rectangle is bounded in constant time; otherwise, the time complexity is $\mathcal{O}(n \log n)$. Results in Table 7 suggest that these instances belong to the first situation. The second situation may occur only if n is extremely large. All results on timing of RSMP include preprocessing (step 1), which is not a time-consuming step; tests show that it does not take more than 1% of the total processing time.

5. Conclusions

A 2D rectangle-packing algorithm is proposed based on a single heuristic policy: the residual space after packing each rectangle should be maximized. To realize the policy, a rectangle is tested for packing in four corners at two orientations of several candidate residual spaces (RSs). The best configuration based on the RSMP policy is chosen as the packing position. The algorithm involves the generation of a series of RSs based on a difference process (DP) and the comparison of the RSs based on their areas. An efficient implementation is described to take advantage of useful information collected along processing so as to minimize the computation.

The performance of RSMP has been compared with state-of-the-art algorithms. Results show that RSMP is comparable to existing algorithms in small-scale problems but is much faster. For large-scale problems, RSMP seems to be the best both in space efficiency and in computational speed. The former is owing to the effective policy, which is well exhibited in the long run, and the latter is owing to the efficient implementation.

Several topics might be meaningful for future research investigation. First, the concept of RS, DP and the principle of RSMP have direct extensions in 3D; efficient implementation of RSMP in 3D packing problems can be investigated. Second, the algorithm proposed in this paper assumes a rectangular bin and RSs are generated by DP; direct generation of RSs from a polygonal bin can be investigated as in some applications, the bins are not rectangular. Third, increasing the number of trial sequences in general leads to better solutions; sequence-generation strategies can be investigated based on RSMP. Last, in irregular item packing problems, it is also reasonable to maximize the residual space during packing; hence, a similar algorithm for packing irregular items can be investigated.

Acknowledgments

This work was supported by the International Design Center of Singapore University of Technology and Design (Grant ID: IDD31200102 and SREP11013).

References

Alvarez-Valdés, R., Parreño, F., & Tamarit, J. M. (2005). A grasp algorithm for constrained two-dimensional non-guillotine cutting problems. *Journal of the Operational Research Society, 56*(4), 414–425.
Aşık, Ö. B., & Özcan, E. (2009). Bidirectional best-fit heuristic for orthogonal rectangular strip packing. *Annals of Operations Research, 172*(1), 405–427.
Baker, B. S., Coffman, E. G., Jr, & Rivest, R. L. (1980). Orthogonal packings in two dimensions. *SIAM Journal on Computing, 9*(4), 846–855.
Beasley, J. E. (1985). An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research, 33*(1), 49–64.
Belov, G., Scheithauer, G., & Mukhacheva, E. (2008). One-dimensional heuristics adapted for two-dimensional rectangular strip packing. *Journal of the Operational Research Society, 59*(6), 823–832.
Bennell, J. A., Lee, L. S., & Potts, C. N. (2013). A genetic algorithm for two-dimensional bin packing with due dates. *International Journal of Production Economics, 145*(2), 547–560.
Berkey, J. O., & Wang, P. Y. (1987). Two-dimensional finite bin-packing algorithms. *Journal of the Operational Research Society, 38*(5), 423–429.
Boschetti, M. A., & Mingozzi, A. (2003). The two-dimensional finite bin packing problem. Part II: New lower and upper bounds. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies, 1*, 135–147.
Burke, E. K., Kendall, G., & Whitwell, G. (2006). *Metaheuristic enhancements of the best-fit heuristic for the orthogonal stock cutting problem. Tech. Rep. NOTTCS-TR-SUB-0605091028-4370*. University of Nottingham.
Burke, E. K., Kendall, G., & Whitwell, G. (2004). A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research, 52*(4), 655–671.
Chazelle, B. (1983). The bottomn-left bin-packing heuristic: An efficient implementation. *IEEE Transactions on Computers, 100*(8), 697–707.
Dagli, C. H., & Poshyanonda, P. (1997). New approaches to nesting rectangular patterns. *Journal of Intelligent Manufacturing, 8*(3), 177–190.
Dell'Amico, M., Martello, S., & Vigo, D. (2002). A lower bound for the non-oriented two-dimensional bin packing problem. *Discrete Applied Mathematics, 118*(1), 13–24.
Dowsland, K. A. (1993). Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research, 68*(3), 389–399.
Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research, 44*(2), 145–159.
Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness* (Vol. 174). Freeman New York.
Gonçalves, J. F. (2007). A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. *European Journal of Operational Research, 183*(3), 1212–1229.
Harwig, J. M., Barnes, J., & Moore, J. T. (2006). An adaptive tabu search approach for 2-dimensional orthogonal packing problems. *Military Operations Research, 11*(2), 5–26.
Hayek, J. E., Moukrim, A., & Negre, S. (2008). New resolution algorithm and pretreatments for the two-dimensional bin-packing problem. *Computers & Operations Research, 35*(10), 3184–3201.
Hopper, E., & Turton, B. C. H. (2001). An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *European Journal of Operational Research, 128*(1), 34–57.
Jakobs, S. (1996). On genetic algorithms for the packing of polygons. *European Journal of Operational Research, 88*(1), 165–181.
Lai, K. K., & Chan, J. W. M. (1997). Developing a simulated annealing algorithm for the cutting stock problem. *Computers & industrial engineering, 32*(1), 115–127.
Leung, S. C. H., & Zhang, D. (2011). A fast layer-based heuristic for non-guillotine strip packing. *Expert Systems with Applications, 38*(10), 13032–13042.

Liu, D., & Teng, H. (1999). An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. *European Journal of Operational Research, 112*(2), 413–420.

Lodi, A., Martello, S., & Vigo, D. (1999). Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing, 11*(4), 345–357.

Lodi, A., Martello, S., & Vigo, D. (2004). Tspack: A unified tabu search code for multi-dimensional bin packing problems. *Annals of Operations Research, 131*, 203–213.

Martello, S., & Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. *Management Science, 44*(3), 388–399.

Özcan, E., Kai, Z., & Drake, J. H. (2013). Bidirectional best-fit heuristic considering compound placement for two dimensional orthogonal rectangular strip packing. *Expert Systems with Applications, 40*, 4035–4043.

Pinto, E., Oliveira, & J. F. (2005). Algorithm based on graphs for the non-guillotinable two-dimensional packing problem. In *Second ESICUP meeting*. Southampton.

Wäscher, G., Haußner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research, 183*(3), 1109–1130.

Wei, L., Oon, W.-C., Zhu, W., & Lim, A. (2011). A skyline heuristic for the 2d rectangular packing and strip packing problems. *European Journal of Operational Research, 215*(2), 337–346.

Zhang, D., Wei, L., Leung, S. C. H., & Chen, Q. (2013). A binary search heuristic algorithm based on randomized local search for the rectangular strip-packing problem. *INFORMS Journal on Computing, 25*(2), 332–345.