Computers & Operations Research 38 (2011) 1035–1044

Contents lists available at ScienceDirect

Computers & Operations Research

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

A squeaky wheel optimisation methodology for two-dimensional strip packing

Edmund K. Burke, Matthew R. Hyde *, Graham Kendall

The University of Nottingham, School of Computer Science, Jubilee Campus, Wollaton Road, Nottingham, UK

A R T I C L E I N F O

Available online 16 October 2010

Keywords:
Cutting stock
Heuristics
Local search
Artificial intelligence

A B S T R A C T

The two-dimensional strip packing problem occurs in industries such as metal, wood, glass, paper, and textiles. The problem involves cutting shapes from a larger stock sheet or roll of material, while minimising waste. This is a well studied problem for which many heuristic methodologies are available in the literature, ranging from the basic 'one-pass' best-fit heuristic, to the state of the art Reactive GRASP and SVC(SubKP) iterative procedures. The contribution of this paper is to present a much simpler but equally competitive iterative packing methodology based on squeaky wheel optimisation. After each complete packing (iteration), a penalty is applied to pieces that directly decreased the solution quality. These penalties inform the packing in the next iteration, so that the offending pieces are packed earlier. This methodology is deterministic and very easy to implement, and can obtain some best results on benchmark instances from the literature.

1. Introduction

Optimisation problems involving cutting stock arise in many industries such as metal, wood, glass, paper, and textiles. These problems generally involve cutting stock sheets of material into smaller shapes, and a typical objective is to minimise the waste. These problems form part of a wider set of cutting and packing problems (see the typologies presented in [1,2]). This paper addresses the two-dimensional orthogonal strip packing problem, where a set of rectangular shapes must be arranged onto a larger rectangle of fixed width and infinite height, while minimising the height of the highest rectangle in the solution, corresponding to a 'guillotine cut' across the width of the sheet. The rectangles must not intersect each other, or exceed the edges of the large rectangle, and the resulting arrangement corresponds to a cutting pattern which will obtain the necessary set of shapes from the stock sheet.

The exact problem that we address in this paper can be defined, by reference to the typology of Wäscher et al. [2], as the two-dimensional orthogonal open dimension problem. We do not use piece rotations, and guillotine cuts are not required, so the problem can also be referred to as the 'OF' sub-type (oriented, free cutting) [3,4].

'Best-fit' is a constructive heuristic for the two-dimensional orthogonal stock cutting problem. A detailed description of the algorithm is given in [5–7], but a brief summary is given here as some of the concepts are used in our squeaky wheel optimisation approach. Best-fit represents the stock sheet as a set of dynamic slots where pieces can be placed. At the start of the packing, there is only one slot extending across the width of the sheet. Given a list of pieces to pack, best-fit iteratively packs a piece into the lowest available slot, and the slot structure is updated after each iteration. The piece that is chosen for the slot is the piece with the largest width, from all of those which can fit. If the lowest available slot is too narrow for any piece, the slot is raised to the level of its lowest neighbour, and they are merged, producing a wider slot. The packing continues in this manner until there are no pieces left to pack.

Burke et al. [6] presented a simulated annealing enhancement to the best-fit heuristic (BF+SA). This was motivated by the observation that the choice of pieces to pack was increasingly limited towards the end of the packing. This often produced 'towers' protruding from the top of the solution, reducing the solution quality. This was initially addressed with a post-processing stage which turned these tall pieces on their side, but there was rarely a good place to put them at that late stage. BF+SA stops packing when m rectangles are left (a parameter which one must tune depending on the size of the instance), and then the rest of the pieces are iteratively packed using a simple bottom-left-fill heuristic, with a simulated annealing algorithm to optimise the piece order.

The current state of the art methodologies are the Reactive GRASP [8] and SVC(SubKP) [9]. The GRASP algorithm involves a constructive phase and a subsequent iterative improvement phase. The method is a complex algorithm with many parameters, which are chosen by hand, and obtains some of the best results in the literature. The SVC (SubKP) uses a constructive heuristic which fills the lowest slot by solving a one-dimensional knapsack problem

* Corresponding author. Tel.: +44 115 84 68376; fax: +44 115 951 4254.
E-mail address: mvh@cs.nott.ac.uk (M.R. Hyde).
URL: http://www.cs.nott.ac.uk/˜mvh/ (M.R. Hyde).

0305-0548/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2010.10.005

with a pseudo-profit for each item. The packing is split into horizontal slices determined by the positions of the pieces, and the profits of the items are based on their size, the size of the slices containing them, and a random scaling variable. In each iteration the profits are updated meaning that the pieces will be packed differently in the next iteration. The algorithm is stochastic. In a given run, probabilities are randomly chosen for changing the order of the pieces in a slot, moving all the pieces to the left or right of the slot, and changing the scaling variable for the pseudo-profits. These probabilities are also modified during the run.

The contribution of this paper is to present a very simple, yet effective, deterministic packing methodology based on 'squeaky wheel optimisation' [10,11]. Squeaky wheel optimisation is a metaheuristic search technique which attempts to identify the most difficult elements of a problem by iteratively constructing a solution, each time analysing the solution to assess which elements are causing problems. In the next iteration, those elements that have been judged to be most difficult are dealt with earlier in the solution construction. BF+SA, GRASP, and SVC (SubKP) are stochastic algorithms, and as such they may require multiple runs to obtain their best solution. In contrast, the squeaky wheel optimisation methodology is deterministic, and therefore requires only one run. While it was not our initial aim, the squeaky wheel optimisation approach is competitive (and sometimes better than) these much more complex algorithms.

2. Related work

This section provides an overview of the relevant literature on cutting and packing problems. More extensive surveys of the field can be found in [1,2,12,13]. Exact methods are not covered here, but recent work on this type of methodology can be found in [14–17].

2.1. Heuristic methods

Baker et al. [18] define a simple class of packing algorithms named 'bottom up, left justified' (BL). Chazelle [19] presents an optimal method for determining the ordered list of points that a piece can be put into. BL type heuristics take, as input, a list of pieces, and the results rely heavily on the pieces being in a 'good' order [18]. Brown et al. [20] show the lower bounds for online algorithms both for pre-ordered piece lists by decreasing height and width, and non pre-ordered lists. Results in [21] have shown that pre-ordering the pieces by decreasing width or decreasing height results in performance increases of between 5% and 10%.

The best-fit heuristic [5] was described briefly in Section 1, and an efficient implementation of best-fit, using appropriate data structures, is presented by Imahori and Yagiura [7], allowing $O(N \log N)$ run time complexity. A simulated annealing enhancement to best-fit can optimise the arrangement of the pieces in the final stage of the packing, where mistakes are more obvious [6]. Zhang et al. [22] use a recursive algorithm, running in $O(N^3)$ time, to create good strip packing solutions, based on a 'divide and conquer' principle.

Two heuristics for the strip cutting problem with sequencing constraints are presented by Rinaldi and Franz [23], and a 'bidirectional' modification of the best-fit algorithm is presented by Asik and Özcan [24], which obtains better results by considering the vertical left hand side of the solution for placement of a piece, not just the lowest horizontal slot.

2.2. Metaheuristic methods

Hopper and Turton [21] provide a survey of the application of metaheuristics to packing problems.

Metaheuristics have been employed to evolve a good ordering of pieces for a simple heuristic to pack [25–27]. However, optimising a piece order for a simple heuristic can be somewhat limited, for example it is shown in [18,9] that, for certain instances, there is no sequence that can be given to the bottom-left-fill (BLF) heuristic [18] that results in the optimal solution. Valenzuela and Wang [28] employ a genetic algorithm with a linear representation of a slicing tree as the chromosome. The slicing tree determines the order that the guillotine cuts are made, and between which pieces. Bortfeldt [4] also implements a genetic algorithm which operates directly on the representations of strip packing solutions, instead of on a piece ordering.

Hopper and Turton [21] compare the performance of several metaheuristic approaches for evolving a piece order, when combined with the BL and BLF algorithms. Better results are obtained when the algorithms are combined with BLF. Lai and Chan [29] and Faina [30] both use a simulated annealing approach to iteratively improve a solution, and achieve good results on small sized problems.

A reactive greedy randomised adaptive search procedure (GRASP) is presented in [8]. The method involves a constructive phase and a subsequent iterative improvement phase. The SVC(SubKP) algorithm [9] is based on an iterative process, repeatedly applying one constructive heuristic, 'SubKP', to the problem, each time updating certain parameters that guide its packing. It is similar to the methodology presented here, but the solution construction and the parameter update step are more complex. In summary, SVC (SubKP) and the reactive GRASP method represent the current state of the art. The squeaky wheel optimisation methodology presented here shows that the quality of results does not need to decrease with a significant decrease in algorithm complexity, and that a deterministic algorithm can achieve consistently good results.

3. Methodology

This section describes our iterative squeaky wheel optimisation packing methodology (SWP), also shown in Algorithm 1. Section 3.1 provides an example of the first six iterations of a run, to further clarify the methodology.

Algorithm 1. Pseudo-code of the squeaky wheel optimisation packing methodology

```
Initialise each piece's penalty value to zero
while elapsed time < time limit do
  while exists pieces not packed do
    Find lowest Slot '$s$'
    if $s$ is too narrow for any piece then
      Raise s to level of lowest neighbour and merge them both
    else
      Find piece with highest penalty, from those that fit into $s$
      Assign this piece to $s$, next to the tallest neighbour
    end if
  end while
  for all Piece '$p$' in current instance do
    if $p$.lower − edge − y − coordinate + $p$.height > instance lowerbound then
      $p$.penalty = $p$.penalty + $p$.height
    end if
  end for
end while
return best solution found
```

SWP takes a very simple constructive heuristic, a reduced version of best-fit, and applies it iteratively. One iteration represents a complete packing using this constructive heuristic. At the start of the run, each piece has a zero 'penalty' value. After each iteration, the penalty value of pieces that directly worsened the solution quality is increased, and the penalty values affect how the constructive heuristic packs the pieces in the next iteration. The pieces judged to have worsened the solution quality are those which exceed the lower bound for the instance.

The constructive heuristic takes the lowest available slot, into which it places the piece with the highest penalty, from those that fit. If two or more pieces that fit into the slot share the same penalty value, then ties are broken by the largest width, and then by the largest height. If the lowest available slot has a width that no remaining piece can fit into, the height of this slot is raised to the level of its lowest neighbour, and the two slots (now with equal height) are merged into one. The process then repeats until all of the pieces are packed.

In preliminary experiments, we found that increasing the piece's penalty value by the size of its height produced the best results. Without penalty values, the constructive heuristic performs identically to the best-fit heuristic with the 'tallest neighbour' placement policy [5]. The addition of penalty values is a minor but fundamental change, which allows the heuristic to learn which pieces are more difficult to allocate, over many iterations.

We test the methodology on two types of instance. The lower bound for the first type corresponds to the zero-waste optimal solution, as these instances are generated from a known optimum. For the second type of instance, the piece sizes were generated randomly, and so the optimal solution generally will contain wasted space. For the second type of instance, we perform experiments using the simple continuous lower bound, and a more sophisticated lower bound from [31].

3.1. Example run

Figs. 1–6 show the first six iterations of a run that finds an optimal solution, with no waste, to the C1P1 instance, in 26 iterations (the instances are explained later in Table 2).

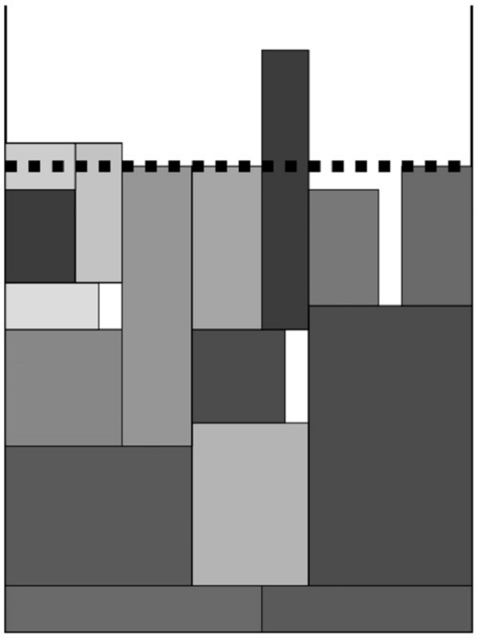
Fig. 1. Iteration 1.

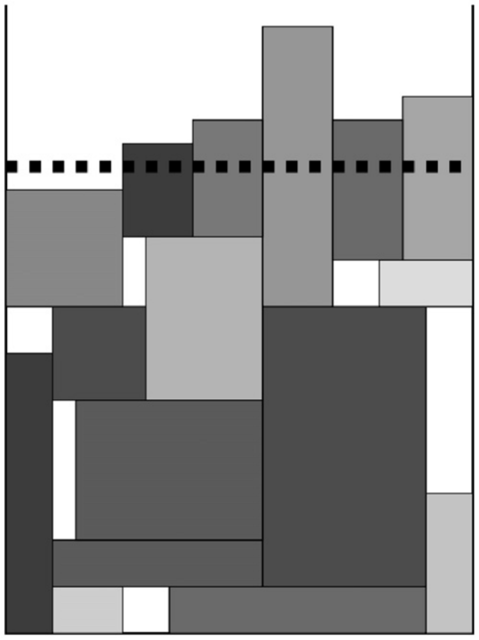
Fig. 2. Iteration 2.

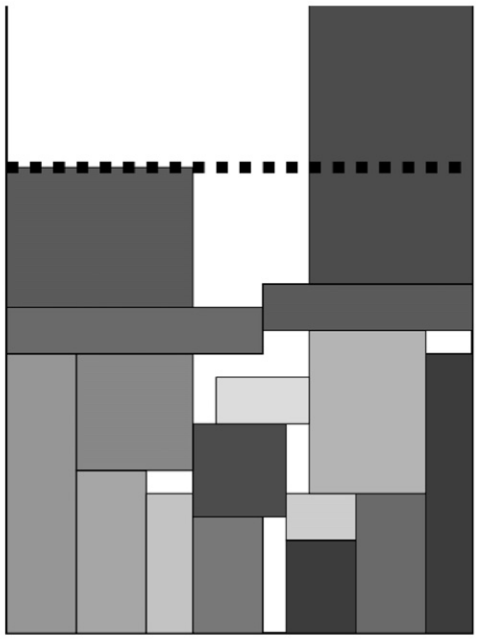
Fig. 3. Iteration 3.

The solution generated in the first iteration is shown in Fig. 1. As the penalty values of each piece are initialised to zero before this iteration, the packing progressed by iterating between finding the lowest slot, selecting the piece that has the largest width from those that fit, and then placing the piece next to its tallest neighbour. Piece rotations are not considered with this methodology. The solution has three pieces which are above the dashed line indicating the optimal solution (or lower bound) for this instance. The heights of these pieces are added to their penalty values. Table 1 shows the penalty values of each piece after each iteration shown in Figs. 1–6. Therefore, the column for the first iteration shows pieces 14, 15 and 16 having penalty values of 2, 12 and 6, respectively (their heights), and the other pieces still have no penalty because they did not exceed the height of the optimal solution.

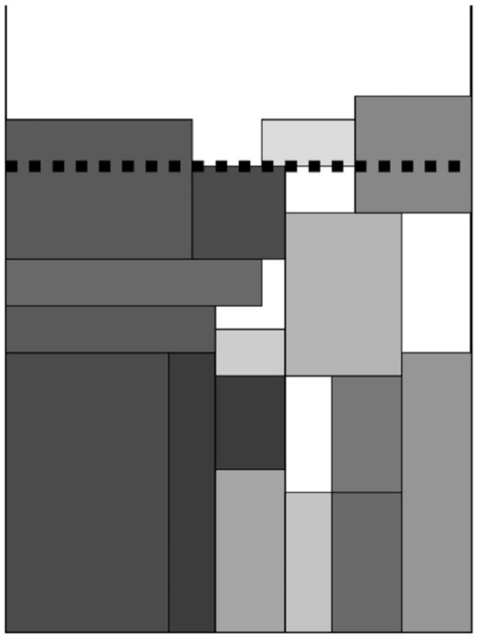
Fig. 4. Iteration 4.

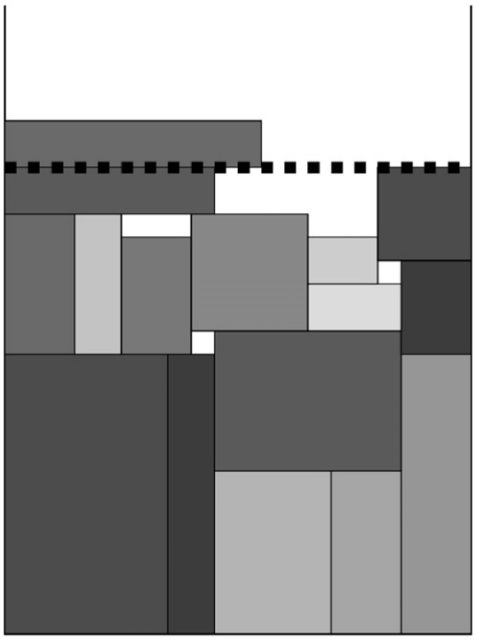
Fig. 6. Iteration 6.

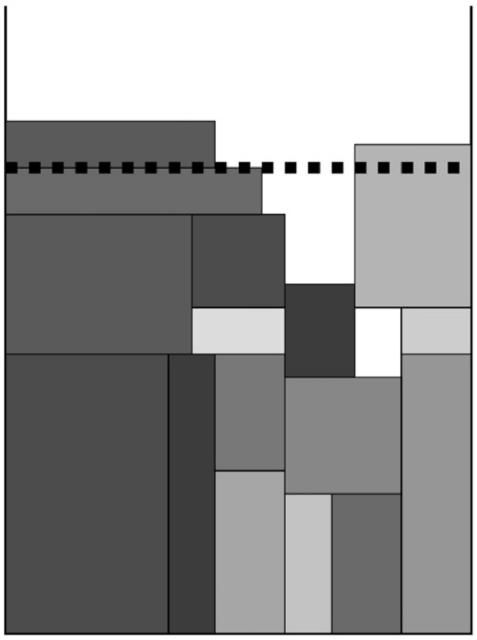
Fig. 5. Iteration 5.

Table 1
Penalty values for each piece after each iteration, shown in Figs. 1–6.

Piece number	Penalty values after each of the first six iterations					
	1	2	3	4	5	6
1	0	0	0	0	0	2
2	0	0	0	0	2	2
3	0	0	0	6	6	6
4	0	0	12	12	12	12
5	0	0	0	0	7	7
6	0	0	0	5	5	5
7	0	0	0	0	0	0
8	0	0	0	2	2	2
9	0	12	12	12	12	12
10	0	7	7	7	7	7
11	0	6	6	6	6	6
12	0	5	5	5	5	5
13	0	4	4	4	4	4
14	2	2	2	2	2	2
15	12	12	12	12	12	12
16	6	6	6	6	6	6

The solution at the end of the second iteration is shown in Fig. 2. The three pieces that received the only penalties in the last iteration have been packed first, two into the lower left of the container and one into the lower right. These positions were chosen for each piece because they were next to their tallest neighbour at the time they were packed. All of the other pieces had the same (zero) penalty value, and so they were packed by breaking ties by the greatest width from those that fit. Five pieces are above the dashed optimal solution line at the end of this iteration, so those pieces receive the penalty values shown in Table 1. Note that the other pieces keep their existing penalty values. No penalty values are ever reduced or reset during a run.

At the end of the third iteration (Fig. 3), there is one piece which exceeds the height of the optimal solution, and so this piece receives a penalty value of 12. The penalty values of all the pieces at this stage can be seen in column 3 of Table 1. Therefore for the next iteration (Fig. 4), the three pieces with a penalty value of 12 have the highest penalty and are therefore packed first. However, it is important to note that the pieces with the highest penalty are not packed first under all circumstances. At each step, only the pieces which can fit into the width of the lowest slot are considered, and from those the piece with the highest penalty is chosen, breaking ties by the largest width. So, if the piece with the highest penalty cannot fit into the lowest slot, then it is not considered at the current step, and must wait until a subsequent step when a larger slot is to be filled.

This example shows the progress of the algorithm until the sixth iteration, after which the iterations continue until an optimal solution is found at iteration 26. This example highlights the range of possible solutions that can be searched with this methodology. The whole solution is modified at each iteration, which contrasts with the methodology of the simulated annealing enhancement to best-fit [6], where just the final section of the solution is modified. Preliminary experiments have shown that better results are obtained with no piece rotations. A piece may be considered difficult to pack in a vertical orientation, but it may not affect the solution as negatively if it is packed in a horizontal orientation, so allowing piece rotations would reduce the relevance of the assigned penalty values.

4. Results

To present the results of our squeaky wheel optimisation packing methodology (SWP), we use the wide variety of problem instances that are publicly available, allowing us to compare with the simulated annealing enhancement to best-fit (BF+SA), the GRASP methodology, and SVC(SubKP). In this paper, we use two classes of benchmark instances from the literature. Class 1 contains instances which were created from known optimal solutions, and class 2 contains instances which were created by randomly generating each piece size. The instances of class 2 do not necessarily have an optimal solution with zero waste.

The results were obtained on the University of Nottingham computing cluster, on AMD Opteron 2.2 GHz single core CPUs. Each run was performed on one CPU, without utilising any parallel processing features. Section 4.1 presents the results on the zero-waste instances (class 1), where the lower bound is set to the optimal solution. Section 4.2 presents the results on the class 2 instances, where the lower bound does not necessarily represent the optimal solution. Both sections show that the SWP methodology is competitive with the current state of the art, and can obtain the best results in the literature for some instances.

4.1. Zero-waste instances

The zero-waste instances employed in this paper are summarised in Table 2. Tables 3 and 4 show the results of the comparison with BF+SA, the GRASP methodology, and SVC(SubKP).

The first comparison to note from Table 3 is that the results are superior to BF+SA. The results are the same or better in almost all of the instances. This is still true when our algorithm is run for 30 s, half the time of the simulated annealing enhancement. The BF+SA methodology works on the assumption that 'mistakes' are only made late in the packing, and so only the pieces that are packed later need to be optimised. The results of SWP suggest that it is possible to benefit from correcting mistakes at any stage, and so it is worth considering changes to the whole solution at each iteration. We argue that SWP is simpler and quicker to implement, yet obtains better results, in less run times.

When compared to the reactive GRASP algorithm of [8], the results are very competitive, and they are even superior on some instances. We would not expect SWP to be comprehensively better, as it is much less complex, and we argue that SWP has a very good ratio of simplicity to quality of results. GRASP is a much more complex algorithm, with many parameters, yet it is very effective, and the results have been shown to be the best in the literature.

Table 2
A summary of the zero waste problem instances used in this paper.

Source	Instance name	Sheet width	Number of rectangles	Optimal height
Burke et al. [5]	N1	40	10	40
	N2	30	20	50
	N3	30	30	50
	N4	80	40	80
	N5	100	50	100
	N6	50	60	100
	N7	80	70	100
	N8	100	80	80
	N9	50	100	150
	N10	70	200	150
	N11	70	200	150
	N12	100	500	300
	N13	640	3152	960
Hopper and Turton [21]	C1P1, C1P2, C1P3	20	16 or 17	20
	C2P1, C2P2, C2P3	40	25	15
	C1P1, C1P2, C1P3	60	28 or 29	30
	C1P1, C1P2, C1P3	60	49	60
	C1P1, C1P2, C1P3	60	72 or 73	90
	C1P1, C1P2, C1P3	80	97	120
	C1P1, C1P2, C1P3	160	196 or 197	240
Valenzuela and Wang [28]	Nice25, Path25	100	25	100
	Nice50, Path50	100	50	100
	Nice100, Path100	100	100	100
	Nice200, Path200	100	200	100
	Nice500, Path500	100	500	100
	Nice1000, Path1000	100	1000	100
Ramesh Babu and Ramesh Babu [27]	P1	1000	50	375
Hopper [32]	N1(P1–P5), T1(P1–P5)	200	17	200
	N2(P1–P5), T2(P1–P5)	200	25	200
	N3(P1–P5), T3(P1–P5)	200	29	200
	N4(P1–P5), T4(P1–P5)	200	49	200
	N5(P1–P5), T5(P1–P5)	200	73	200
	N6(P1–P5), T6(P1–P5)	200	97	200
	N7(P1–P5)	200	197	200
	T7(P1–P5)	200	199	200

Table 3
The results of the proposed squeaky wheel optimisation packing methodology, compared to the Reactive GRASP approach [8], best-fit with simulated annealing enhancement [6], and the basic best-fit heuristic [5].

Instance name	Best-fit	Best fit+SA (60s)	Reactive GRASP (60 s)		Squeaky wheel (60 s)	Time to find best solution	
			Mean	Best		Iterations	Seconds
N1	45	40	40	40	40	2	< 0.1
N2	53	50	50	50	50	2888	0.5
N3	52	51	51	51	50	1259185	41.9
N4	86	82	81	81	81	16	< 0.1
N5	105	103	102	102	101	558000	41.8
N6	102	102	101	101	101	4065	0.8
N7	108	104	101	101	101	55	0.2
N8	83	82	81	81	81	52446	8.3
N9	152	152	151	151	151	14819	3.6
N10	152	152	151	151	151	329	0.9
N11	153	153	151	151	151	81	0.7
N12	306	306	303.2	303	304	128	0.9
N13	964	964	963	963	966	65	9.0
C1P1	21	20	20	20	20	26	< 0.1
C1P2	22	20	20	20	21	11	0.01
C1P3	24	20	20	20	20	2	< 0.1
C2P1	16	16	15	15	16	12	< 0.1
C2P2	16	16	15	15	15	1312	0.5
C2P3	16	16	15	15	15	17	< 0.1
C3P1	32	31	30	30	30	6360	0.7
C3P2	34	31	31	31	31	22	< 0.1
C3P3	33	31	30	30	30	7168	0.6
C4P1	63	61	61	61	61	2360	0.4
C4P2	62	61	61	61	61	774	0.5
C4P3	62	61	61	61	61	290	0.6
C5P1	93	91	91	91	91	13	< 0.1
C5P2	92	91	91	91	91	1922	0.9
C5P3	93	92	91	91	91	8594	1.4
C6P1	123	122	121.9	121	122	137	0.5
C6P2	122	121	121.9	121	121	2114	1.4
C6P3	124	122	121.9	121	122	544	0.7
C7P1	246	244	244	244	243	22142	15.2
C7P2	244	244	242.9	242	242	2352	2.3
C7P3	245	245	243	243	243	1632	1.8
Nice25	108	104	103.9	103.7	103.7	388299	12.3
Nice50	109.7	104.4	104.7	104.6	104.9	253034	21.7
Nice100	107.9	105	104.5	104	104.6	51117	14.1
Nice200	106.9	104.7	103.8	103.6	103.8	50353	48.3
Nice500	103.4	103.5	102.4	102.2	103.3	9607	49.2
Nice1000	103.8	103.8	102.3	102.2	102.9	857	19.9
Path25	110.2	103.1	104.2	104.2	106.9	4106	0.8
Path50	113.6	103.4	101.9	101.8	101.7	727801	58.4
Path100	106.7	103	102.7	102.6	102.9	19687	6.6
Path200	104.1	103.4	102.3	102	102.0	31767	30.7
Path500	103.8	103.5	103.2	103.1	103.2	4829	24.6
Path1000	103.1	102.9	102.7	102.5	102.8	2904	59.8
P1	400	400	375	375	400	2	< 0.1

Table 4 shows a comparison between SWP and the SVC(SubKP) methodology of Belov et al. [9]. The results are displayed in a new table because Belov et al. [9] present results on different sets of instances, and they are presented as averages of each problem class. The results of the GRASP algorithm on some of these instances are also repeated in Table 4 for comparison. The results of our very simple SWP algorithm are very competitive with SVC(SubKP), and they are better in three instance classes. GRASP is slightly worse than SVC(SubKP) and SWP on these instances, so SWP sits between both of these successful algorithms. SVC(SubKP) is also an iterative algorithm which updates values for each piece at the end of each iteration. These values are then used in the next iteration to change the behaviour of the constructive heuristic SubKP. Both the constructive packing heuristic and the penalty value calculation in SWP are simpler and easier to implement. SWP is a deterministic algorithm, while GRASP and SVC(SubKP) have stochastic elements, therefore they may need a number of runs to obtain their best result, while SWP needs only one run.

4.2. *Randomly generated instances*

Section 4.1 discussed the results obtained on the instances with known zero-waste optimal solutions. If a piece exceeds the line representing the height of this optimal solution, then it is clear that it has worsened the solution quality. For instances without a known

Table 4
The results of the proposed squeaky wheel optimisation packing methodology, compared to SVC(SubKP) [9] and GRASP [8], on the waste free instances of [32].

Instance class	SVC(SubKP) (50 s)	Reactive GRASP (60 s)		Squeaky wheel (60 s)	Average time to best solution	
		Mean	Best		Iterations	Seconds
C1 (P1–P3)	20.3	20	20	20.3	13	< 0.1
C2 (P1–P3)	15	15	15	15.3	447	0.2
C3 (P1–P3)	30.3	30.3	30.3	30.3	4516.667	0.5
C4 (P1–P3)	61	61	61	61	1141.333	0.5
C5 (P1–P3)	91	91	91	91	3509.667	0.8
C6 (P1–P3)	121	121.9	121	121.7	931.6667	0.9
C7 (P1–P3)	242	243.3	243	242.7	8708.667	6.4
N1 (P1–P5)	206.6		200	209.4	241024.6	5.6
N2 (P1–P5)	206.8		206.4	206.6	210911.8	7.0
N3 (P1–P5)	207		207.4	207	89043	4.3
N4 (P1–P5)	205		206	204.8	215918.4	16.5
N5 (P1–P5)	204.2		204.8	204.6	123577.2	17.2
N6 (P1–P5)	203.4		204.2	203.8	63698.8	13.6
N7 (P1–P5)	202		203	202.4	10556.6	7.8
T1 (P1–P5)	201.8		201.8	210.2	101591.8	2.6
T2 (P1–P5)	207		206.6	207	102778	3.6
T3 (P1–P5)	206.6		207.2	207	288913.6	11.0
T4 (P1–P5)	205		206	204.8	226342	17.5
T5 (P1–P5)	204.2		205.2	204.2	163459	22.2
T6 (P1–P5)	203.2		204.4	203.4	78916.4	16.8
T7 (P1–P5)	202		202.6	203.2	1938.8	2.0

Table 5
A summary of the 500 problem instances used in this paper which are not created from a known optimal solution.

Sources	Class	Number of instances	Sheet width	Number of rectangles
Martello and Vigo [33] & Berkey and Wang [34]	C1	5×10	10	20, 40, 60, 80, or 100
	C2	5×10	30	20, 40, 60, 80, or 100
	C3	5×10	40	20, 40, 60, 80, or 100
	C4	5×10	100	20, 40, 60, 80, or 100
	C5	5×10	100	20, 40, 60, 80, or 100
	C6	5×10	300	20, 40, 60, 80, or 100
	C7	5×10	100	20, 40, 60, 80, or 100
	C8	5×10	100	20, 40, 60, 80, or 100
	C9	5×10	100	20, 40, 60, 80, or 100
	C10	5×10	100	20, 40, 60, 80, or 100

optimal solution, the ideal height of this lower bound line is not so clear. In this section, we investigate if the SWP methodology can still be successfully employed to solve such instances, and also whether it makes a difference how the lower bound is calculated. Table 5 summarises the benchmark instances employed for this section, which do not necessarily have a zero-waste optimal solution. These instances are widely used in the literature, and are explained well in [3].

Table 6 presents the results of the SWP methodology on the instances which are not created from a known optimal solution. The results are compared to those of the SVC(SubKP) [9] and GRASP [8] algorithms. In Table 6, three sets of lower bounds are displayed. LB1 is the continuous lower bound, obtained by dividing the sum of the piece areas by the width of the strip. LB2 is the more sophisticated lower bound, presented in [31].

The results of SWP are presented with the lower bound line set as both LB1 and LB2. The results using LB2 are generally better, although the difference is small. LB2 is quick to calculate, although it takes slightly more time to implement, so we conclude that LB2 should be implemented for this methodology if possible. However, if simplicity of implementation is paramount, then LB1 can be employed to obtain very similar results.

The table shows that the results of SWP are competitive with the best in the literature. In summary, GRASP and SVC(SubKP) generally obtain better results on the smaller instances of classes C1 and C2, while the results are approximately equal on classes C7 and C8. On classes C4 and C6, SWP obtains results which are worse than SVC(SubKP), but better than GRASP. SWP obtains the best results in the literature on classes C3, C5, C8, and C10.

4.3. Example packings

Figs. 7–9 show the results obtained on instances T4-2, C7P2 and Path200, respectively. They represent some of the best results obtained by SWP. Also, to our knowledge, no previous methodology from the literature has reported the optimal solution of 50 for the N3 instance.

5. Conclusions

This paper has presented a squeaky wheel optimisation packing methodology (SWP), which obtains superior results to a previously published simulated annealing methodology (BF+SA) [6], and it is competitive with the state of the art Reactive GRASP [8] and SVC(SubKP) [9]. SWP is shown to be capable of generating the best results in the literature on some benchmark instances.

SWP is very easy to implement, particularly when compared to other modern strip packing methodologies. In addition to its straightforward constructive heuristic, SWP consists of just a 'while' loop which iterates until a time limit is reached, storage

Table 6
The results of the proposed squeaky wheel optimisation packing methodology, compared to SVC(SubKP) [9] and GRASP [8], on the instances of [33], and [34].

	n	Lower bounds		Reactive GRASP	SVC(SubKP)	Squeaky wheel (SWO)		Ave time to best solution (SWO)	
		LB1	LB2			LB1	LB2	Iterations	Sec.
C1	20	58.2	59.9	61.3	61.4	61.5	61.4	447.4	0.1
	40	116.4	120.8	121.9	122.0	122.0	122.0	525.8	0.2
	60	179.6	187.4	188.7	188.5	188.7	188.6	12265.4	1.5
	80	248.5	262.2	262.9	262.6	262.8	262.8	762.1	0.3
	100	300.2	304.4	305.6	304.9	305.4	305.3	557.1	0.3
C2	20	19.7	19.7	19.8	19.8	20.0	20.0	146.5	0.1
	40	39.1	39.1	39.1	39.1	39.1	39.1	4025.1	0.5
	60	60.1	60.1	60.2	60.1	60.2	60.2	20801.3	2.2
	80	83.2	83.2	83.2	83.2	83.3	83.3	3663.4	1.1
	100	100.5	100.5	100.5	100.5	100.6	100.6	918.7	0.7
C3	20	152.8	157.4	163.5	164.6	164.2	164.2	3648.5	0.4
	40	308.8	327.6	333.8	333.9	333.7	333.4	106799.4	6.3
	60	481.6	499.2	506.6	506.8	506.1	506.4	65342.4	7.1
	80	667.1	701.7	710.0	709.8	709.8	709.6	23841.4	4.1
	100	804.0	832.7	840.2	839.5	839.0	838.9	36297.4	8.2
C4	20	61.4	61.4	63.4	63.9	63.5	63.5	481596.6	10.4
	40	123.9	123.9	126.3	125.9	125.6	125.6	344949.4	19.2
	60	193.0	193.0	196.7	195.6	196.0	196.0	200560.2	19.8
	80	267.2	267.2	272.2	270.4	271.3	271.3	154038.3	24.2
	100	322.0	322.0	327.3	325.4	326.8	326.8	43644.8	10.7
C5	20	478.8	512.2	533.9	537.4	537.3	537.8	12352.6	0.4
	40	969.8	1053.8	1074.7	1076.5	1073.7	1073.7	1498.4	0.5
	60	1514.6	1613.4	1645.9	1648.1	1642.7	1642.9	110801.1	11.9
	80	2097.6	2268.4	2290.4	2288.5	2287.7	2287.8	50550.0	8.8
	100	2529.9	2617.4	2652.0	2652.1	2646.1	2645.0	53941.9	13.9
C6	20	159.9	159.9	167.3	168.6	167.6	167.6	389671.6	9.2
	40	323.5	323.5	333.6	332.2	332.0	332.0	333381.2	19.4
	60	505.1	505.1	520.6	516.9	517.7	517.7	198774.4	22.0
	80	699.7	699.7	718.9	714.0	716.2	716.2	143611.1	24.4
	100	843.8	843.8	865.4	860.5	862.9	862.9	112493.3	28.7
C7	20	424.6	480.0	501.9	501.9	501.9	501.9	9.0	0.0
	40	914.4	1039.5	1059.0	1059.9	1059.4	1059.0	15042.7	1.1
	60	1341.1	1503.5	1529.6	1530.0	1529.6	1529.6	51.7	0.1
	80	1935.3	2195.5	2222.2	2222.1	2222.1	2222.1	498.6	0.5
	100	2334.8	2621.2	2645.2	2644.0	2645.9	2644.4	9106.4	3.3
C8	20	434.4	434.6	458.0	461.5	458.6	458.7	165141.5	4.5
	40	921.6	922.0	954.4	956.1	949.3	949.3	385059.9	22.3
	60	1360.9	1360.9	1405.9	1400.8	1400.7	1400.7	249375.8	26.0
	80	1909.2	1909.3	1973.6	1964.8	1956.5	1957.0	201587.4	32.4
	100	2362.5	2362.8	2439.5	2423.1	2416.7	2416.6	89437.0	21.8
C9	20	891.3	1103.5	1106.8	1106.8	1106.8	1106.8	1.0	0.0
	40	1757.1	2179.2	2190.6	2190.6	2191.0	2190.6	87.0	0.0
	60	2718.6	3394.0	3410.4	3410.4	3410.4	3410.4	1.0	0.0
	80	3652.8	4563.9	4588.1	4588.1	4588.1	4588.1	1.0	0.0
	100	4453.2	5415.6	5434.9	5434.9	5434.9	5434.9	1.0	0.0
C10	20	327.7	337.4	350.5	351.4	351.8	351.6	169205.9	4.0
	40	632.6	642.8	664.5	667.2	663.0	663.4	275649.0	16.2
	60	899.6	911.1	935.5	936.0	931.8	932.0	124552.0	13.6
	80	1172.3	1177.6	1209.7	1211.2	1204.2	1204.4	165432.3	26.7
	100	1470.9	1476.5	1515.1	1513.9	1504.4	1504.8	86254.5	21.7

of a dynamic penalty value for each piece, and a check at the end of each iteration to assign these penalties. All of the state of the art iterative packing procedures (BF+SA, GRASP, and SVC(SubKP)) are more complex, and this paper shows that the quality of results does not need to decrease when the algorithm complexity is decreased.

BF+SA performs the majority of the packing with the best-fit heuristic, and then optimises the final stage of the packing with an iterative procedure. BF+SA, therefore, contains an implicit assumption that 'mistakes' are not made early in the packing by assigning a piece to a place that will cause problems later on, and it assumes that mistakes need to be rectified only at the end of the packing to obtain the best solutions. The SWP methodology may obtain better results because it considers the whole solution for improvement in each iteration, and it assumes that a mistake could have been made at any point in the packing.

The results are also surprisingly competitive with those of the reactive GRASP methodology [8] and the SVC(SubKP) algorithm [9]. These are complex but very effective methodologies, which have previously been shown to obtain the best results in the literature for the strip packing problem. The GRASP algorithm in particular has many parameters and sub-algorithms, and is certainly more time consuming to implement than SWP. As GRASP and SVC(SubKP) are stochastic algorithms, they may require multiple runs to obtain the best result. In contrast, SWP is fully deterministic, and so only

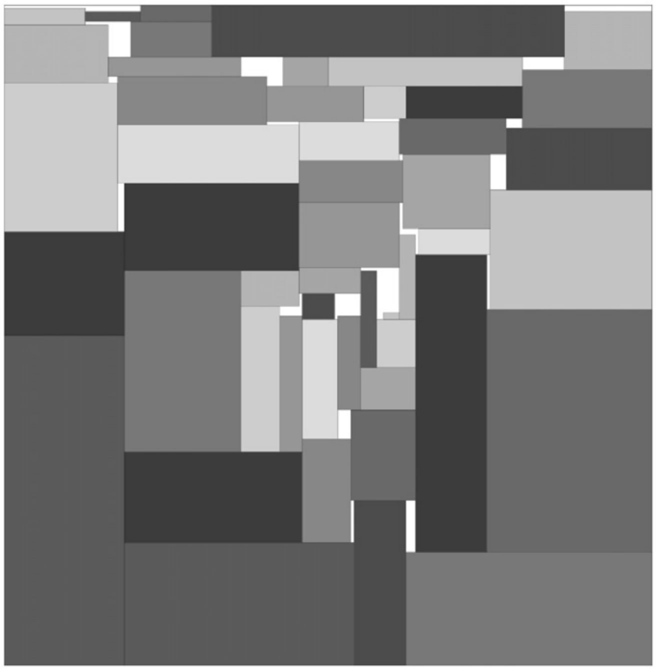
Fig. 7. The solution found for the second instance of the 'T4' class, with a height of 204.

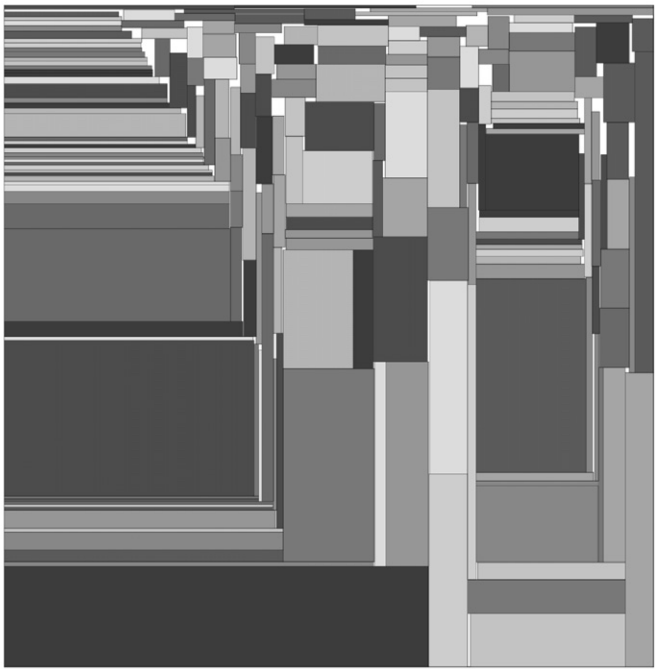
Fig. 9. The solution found for instance 'Path200' with a height of 101.97.

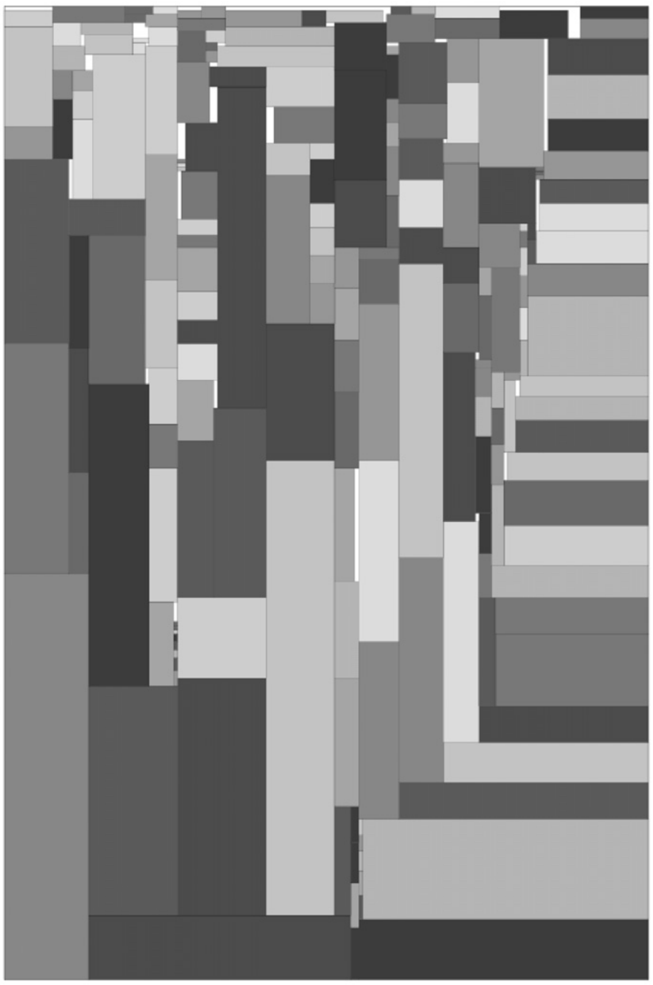
Fig. 8. Solution found for instance 'C7P2' with a height of 242.

requires one run. SWP also obtains the optimal solution for the N3 instance, which, to our knowledge, has not been reported in the literature before. To summarise the comparison with the GRASP and SVC(SubKP) methodologies, we obtain very competitive results, with a more straightforward, easier to implement, deterministic method.

In some applications, it is imperative to obtain results as close to optimal as possible. In this problem domain, if this is the case, then there is a strong argument for implementing either the reactive GRASP or the SVC(SubKP) methodologies. However, their complexity means there is a cost associated with these implementations, and the cost can outweigh the benefit to an organisation when simpler methodologies are available. This is the reason that simple but powerful constructive heuristics such as best-fit have a niche in operational research. The literature reflects the practical necessity of heuristics that fill this niche. SWP sits in the middle of these two extremes, as it uses a constructive heuristic which is simpler than best-fit, but applies it iteratively. The results of SWP are very close to, and sometimes better than, the best in the literature.

Acknowledgements

This work was supported by ESPRC grant reference EP/C523385/1. The genetic programming was implemented using the ECJ package, a Java based evolutionary computation research system, which can be found at http://www.cs.gmu.edu/~eclab/projects/ecj/.

References

[1] Dyckhoff H. A typology of cutting and packing problems. European Journal of Operational Research 1990;44(2):145–59.
[2] Wäscher G, Haußner H, Schumann H. An improved typology of cutting and packing problems. European Journal of Operational Research 2007;183(3): 1109–30.
[3] Lodi A, Martello S, Vigo D. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. INFORMS Journal on Computing 1999;11(4):345–57. ISSN 1526-5528.
[4] Bortfeldt A. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. European Journal of Operational Research 2006;172(3): 814–37.

[5] Burke E, Kendall G, Whitwell G. A new placement heuristic for the orthogonal stock-cutting problem. Operations Research 2004;55(4):655–71.
[6] Burke E, Kendall G, Whitwell G. A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock cutting problem. INFORMS Journal on Computing 2009;21(3):505–16.
[7] Imahori S, Yagiura M. The best-fit heuristic for the rectangular strip packing problem: an efficient implementation and the worst-case approximation ratio. Computers and Operations Research 2009;37(2):325–33.
[8] Alvarez-Valdes R, Parreno F, Tamarit JM. Reactive GRASP for the strip-packing problem. Computers and Operations Research 2008;35(4):1065–83.
[9] Belov G, Scheithauer G, Mukhacheva EA. One-dimensional heuristics adapted for two-dimensional rectangular strip packing. Journal of the Operational Research Society 2008;59(6):823–32.
[10] Joslin DE, Clements DP. "Squeaky Wheel" optimisation. Journal of Artificial Intelligence Research 1999;10:353–73.
[11] Aickelin U, Burke EK, Li J. An evolutionary squeaky wheel optimization approach to personnel scheduling. IEEE Transactions on Evolutionary Computation 2009;13(2):433–43.
[12] Dowsland KA, Dowsland W. Packing problems. European Journal of Operational Research 1992;56(1):2–14.
[13] Lodi A, Martello S, Monaci M. Two-dimensional packing problems: a survey. European Journal of Operational Research 2002;141(2):241–52.
[14] Clautiaux F, Jouglet A, Carlier J, Moukrim A. A new constraint programming approach for the orthogonal packing problem. Computers and Operations Research 2008;35(3):944–59.
[15] Kenmochi M, Imamichi T, Nonobe K, Yagiura M, Nagamochi H. Exact algorithms for the two-dimensional strip packing problem with and without rotations. European Journal of Operational Research 2009;198(1):73–83.
[16] Macedo R, Alves C, Valerio de Carvalho JM. Arc-flow model for the two-dimensional guillotine cutting stock problem, Computers and Operations Research 2010;37(6)991–1001.
[17] Alvarez-Valdes R, Parreno F, Tamarit JM. A branch and bound algorithm for the strip packing problem. OR Spectrum 2009;31(2):431–59.
[18] Baker BS, Coffman Jr. EG, Rivest RL. Orthogonal packings in two dimensions. SIAM Journal on Computing 1980;9(4):846–55.
[19] Chazelle B. The bottom-left bin packing heuristic: an efficient implementation. IEEE Transactions on Computers 1983;32(8):697–707.
[20] Brown DJ, Baker BS, Katseff HP. Lower bounds for on-line two-dimensional packing algorithms. Acta Informatica 1982;18(2):1982.
[21] Hopper E, Turton BCH. An Empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. European Journal of Operational Research 2001;128(1):34–57.
[22] Zhang D, Kang Y, Deng A. A new heuristic recursive algorithm for the strip rectangular packing problem. Computers and Operations Research 2006;33(8): 2209–17.
[23] Rinaldi F, Franz A. A two-dimensional strip cutting problem with sequencing constraint. European Journal of Operational Research 2007;183(3):1371–84.
[24] Asik OB, Özcan E. Bidirectional best-fit approach for orthogonal rectangular strip packing. Annals of Operations Research 2009;172:405–27.
[25] Jakobs S. On genetic algorithms for the packing of polygons. European Journal of Operational Research 1996;88(1):165–81.
[26] Liu D, Teng H. An improved BL-algorithm for genetic algorithms of the orthogonal packing of rectangles. European Journal of Operational Research 1999;112(2):413–9.
[27] Ramesh Babu A, Ramesh Babu N. Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms. International Journal of Production Research 1999;37(7):1625–43.
[28] Valenzuela CL, Wang PY. Heuristics for large strip packing problems with guillotine patterns: an empirical study. In: Proceedings of the metaheuristics international conference 2001 (MIC'01), University of Porto, Porto, Portugal, 2001. p. 417–21.
[29] Lai KK, Chan JWM. Developing a simulated annealing algorithm for the cutting stock problem. Computers and Industrial Engineering 1997;32(1): 115–27.
[30] Faina L. An application of simulated annealing to the cutting stock problem. European Journal of Operational Research 1999;114(3):542–56.
[31] Martello S, Monaci M, Vigo D. An exact approach to the strip-packing problem. INFORMS Journal on Computing 2003;15(3):310–9.
[32] Hopper E. Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods. PhD Thesis, Cardiff University; 2000.
[33] Martello S, Vigo D. Exact solution of the two-dimensional finite bin packing problem. Management Science 1998;44(3):388–99.
[34] Berkey JO, Wang PY. Two-dimensional finite bin-packing algorithms. Journal of the Operational Research Society 1987;38(5):423–9.