Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 899 – 908

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

Solving the 2D bin packing problem by means of a hybrid evolutionary algorithm

Christian Blum[a,b,*], Verena Schmid[c,d]

[a]*Department of Computer Science and Artificial Intelligence*
University of the Basque Country, San Sebastian, Spain
[b]*IKERBASQUE, Basque Foundation for Science, Bilbao, Spain*
[c]*Department of Business Administration, Universität Wien, Vienna, Austria*
[d]*Departamento de Ingeniería Industrial, Universidad de Los Andes, Bogotá, Colombia*

Abstract

Combinatorial optimization problems dealing with 2D bin packing find applications, for example, in the context of transportation/warehousing and for the cutting of glass, wood, and metal. In this work we consider the oriented 2D bin packing problem under free guillotine cutting, a problem in which a set of oriented rectangular items is given which must be packed into a minimum number of bins of equal size. Our algorithm proposal to tackle this problem concerns an evolutionary algorithm that makes heavy use of a randomized one-pass heuristic for constructing solutions. The results of the proposed algorithm are compared to some of the best approaches from the literature. This comparison shows that our algorithms is very competitive to state-of-the-art approaches. In particular, the optimal solutions to four previously unsolved instances were found.

Keywords: 2D bin packing, free guillotine cutting, evolutionary algorithm, hybrid computational system

1. Introduction

In this work we propose a hybrid evolutionary algorithm for solving the 2D bin packing problem in which the items have a fixed orientation and free guillotine cutting is allowed. According to Lodi et. al [1] this problem is denoted in the literature as 2BP|O|F. Henceforth we will refer to this problem simply by *2BP*. Concerning the complexity, note that the 2BP problem is an NP-hard combinatorial optimization problem (see [2]). In general, bin packing problems have been extensively studied in the past. The main reason for their popularity is a large number of real-world applications. Real world applications for the 2BP include, for example, cutting glass, wood, or metal and packing in the context of transportation or warehousing (see [3, 4]).

As mentioned already above, the 2BP is tackled in this work by a hybrid evolutionary algorithm. This algorithm can be attributed to the class of hybrid metaheuristics, a field of research which deals with combinations of metaheuristics, such as evolutionary algorithms and tabu search, with other techniques for optimization, such as heuristics and exact methods. It has been shown that such approaches may strongly benefit from a synergy

*Corresponding author
E-mail address: christian.blum@acm.org.

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.255

between the combined algorithmic approaches. For more information on hybrid metaheuristics we refer the interested reader to [5]. The hybrid aspect of the evolutionary algorithm proposed in this work concerns the representation of solutions. In fact, instead of working directly on solutions to the problem, the search space of the algorithm consists of all permutations of the items belonging to a problem instance. In order to determine the objective function value of a permutation, the LGFi heuristic proposed in [6] is used. This heuristic takes any permutation of all the items as input and produces a solution to the corresponding 2BP problem instance in the specific way which is outlined in Section 4.1. The experimental results show that the proposed algorithm compares favorably to current state-of-the-art approaches. In fact, summing up the number of used bins concerning all considered 500 problem instances, the evolutionary algorithm reaches a value of 7239, which is the best value reached by any algorithm that has been proposed for this problem.

The organization of the paper is as follows. In Section 2 we provide an overview of related work. Next, in Section 3 we provide a technical description of the tackled problem. The proposed algorithm is then presented in Section 4. Finally, an experimental evaluation is provided in Section 5, while conclusions and an outlook to the future are given in Section 6.

2. Related Work

In the following we focus on rather recent work on (meta-)heuristics for the 2BP problem. A good overview on earlier work may be found in [7, 8, 9, 10].

Heuristics. The literature mainly distinguishes between *one-phase approaches* and *two-phase approaches*. One-phase algorithms pack the items directly into bins, whereas two-phase algorithms first pack the items into levels, that is, horizontal packings of items that do not exceed the bin width, and then stack these levels into bins. Well known *level-packing algorithms* are Next-Fit Decreasing Height (NFDH), First-Fit Decreasing Height (FFDH) and Best-Fit Decreasing Height (BFDH) [11]. These strategies were originally developed for the one-dimensional bin packing problem, but have later been adapted to strip packing problems and to the two-dimensional case. All three heuristics require the items to be sorted by non-increasing height, which represents the order in which they are packed. Moreover, they pack the items into one bin of infinite height.

Among the most important *one-phase approaches* are Alternate Direction (AD) [1], Bottom-Left Fill (BLF) [12], Improved Lowest Gap Fill (LGFi) [6] and Touching Perimeter (TP) [1]. Concerning *two-phase approaches* we distinguish between Hybrid Next-Fit (HNF) (see [13]) which is based on NFDH, Hybrid First-Fit (HFF) [14] which is based on FFDH and Finite Best-Strip (FBS) [15]—also sometimes referred to as Hybrid Best-Fit—which is based on BFDH. Another example concerns Knapsack Packing (KP) [1]. Finally, Floor Ceiling (FC) [1] can be seen as an improvement over FBS.

The best heuristic for the 2BP which is currently available (labelled SCH) is based on solving a set-covering formulation of the problem [16] by means of column generation. In the first phase, a rather small subset of all possible columns is generated by using greedy procedures and fast constructive heuristic algorithms from the literature. In the second phase, the resulting set-covering instance is solved by means of a Lagrangian-based heuristic.

In addition, some heuristics developed for three-dimensional packing can sometimes easily be applied to the 2BP. An example is the extreme point based heuristic from [17]. This heuristic uses extreme points to determine all points in the bin where items can be placed. Extreme points can either be corners of the already placed items or points generated by the extended edges of the placed items. These points are updated every time an item is placed into the bin. For placing the items a modified version of BFDH is used.

Metaheuristics. Apart from heuristics, more recently researchers also explored the application of metaheuristics to the 2BP problem. The earliest metaheuristic developed for the 2BP was *tabu search* (TS) [18, 7]. Moreover, different researchers have applied *guided local search* (GLS) [19] and *weight annealing* (WA) [20]. A rather simple metaheuristic, labeled HBP, based on a greedy heuristic has been proposed in [21]. HBP assigns a score

to each item. Then, for the construction of a solution, the items are considered according to non-increasing values of the scores. After the construction of a solution the scores are updated using a certain criterion. This procedure is iterated until a pre-defined stopping criterion is met. Finally, the currently best-performing metaheuristic is a hybrid between a greedy randomized adaptive search procedure (GRASP) and variable neighborhood descent (VND) [22]. The solution construction phase of GRASP is hereby based on a maximal-space heuristic from the field of container loading.

3. The 2D Bin Packing Problem

A technical description of the considered problem is given in the following by means of an ILP model from [23].[1] In this model, let $\mathcal{I} = \{1, \ldots, n\}$ be the set of items that have to be packed into a minimum number of equal-sized bins of width W and height H. Each item $i \in \mathcal{I}$ is characterized by its width w_i and height h_i. In the context of this paper note that W and H as well as w_i and h_i (for all $i \in \mathcal{I}$) are integer values, which is sufficient for solving the considered problem instances. The ILP model consists of the following variables. For each $i \in \mathcal{I}$

1. the value of variable m_i indicates the number of the bin in which item i is placed;
2. the values of variables x_i and y_i indicate the lower left coordinates of the position of item i in bin m_i.

Moreover, for each pair $i, j \in \mathcal{I}$ (such that $i \neq j$)

1. the binary variable l_{ij} is set to 1 iff item i is located *left* of item j;
2. the binary variable b_{ij} is set to 1 iff item i is located *below* item j;
3. the binary variable p_{ij} assumes value 1 iff $m_i < m_j$.

Finally, variable v indicates the number of bins in use. Using these variables the following ILP solves the 2D bin packing problem without item rotation and under free guillotine cutting.

$$\textbf{min } v \tag{1}$$

subject to:

$$l_{ij} + l_{ji} + b_{ij} + b_{ji} + p_{ij} + p_{ji} \geq 1 \quad \text{for } i, j \in \mathcal{I}, i < j \tag{2}$$

$$x_i - x_j + W l_{ij} \leq W - w_i \quad \text{for } i, j \in \mathcal{I} \tag{3}$$

$$y_i - y_j + H b_{ij} \leq H - h_i \quad \text{for } i, j \in \mathcal{I} \tag{4}$$

$$m_i - m_j + n p_{ij} \leq n - 1 \quad \text{for } i, j \in \mathcal{I} \tag{5}$$

$$x_i \leq W - w_i \quad \text{for } i \in \mathcal{I} \tag{6}$$

$$y_i \leq H - h_i \quad \text{for } i \in \mathcal{I} \tag{7}$$

$$m_i \leq v \quad \text{for } i \in \mathcal{I} \tag{8}$$

$$1 \leq m_i \quad \text{for } i \in \mathcal{I} \tag{9}$$

$$m_i \leq i \quad \text{for } i \in \mathcal{I} \tag{10}$$

$$l_{ij}, b_{ij}, p_{ij} \in \{0, 1\} \quad \text{for } i, j \in \mathcal{I}, i \neq j$$

$$x_i, y_i, m_i, v \in \mathbb{N}$$

Hereby, constraints (2), (3), (4), (6) and (7) take care that no overlaps occur between two items and/or between items and the borders of the bins. Constraints (5) are responsible for a correct setting of the p_{ij}-variables. Furthermore, constraints (8) and (9) limit the bin numbers to $[1, v]$. Finally, constraints (10) are symmetry breaking constraints that facilitate the work of the MIP solver.

4. The Proposed Algorithm

As mentioned already in the introduction, the proposed evolutionary algorithm is strongly based on heuristic LGFi, which was described by Wong and Lee in [6]. In the following we first outline the way in which the LGFi heuristic works. Afterwards, the evolutionary algorithm is described in detail.

[1] Note that an alternative model was presented in [24].

Algorithm 1 Evolutionary Algorithm for the 2BP (EA-LGFi)

1: **input:** values for parameters p_{size}, c_{rate}, κ and δ
2: $P :=$ GenerateInitialPopulation$(p_{\text{size}}, \kappa)$
3: **while** stopping criterion not met **do**
4: $\quad P' :=$ Crossover$(P, c_{\text{rate}}, \delta)$
5: $\quad P :=$ AddNewSolutions$(P', p_{\text{size}}, \kappa)$
6: **end while**
7: **output:** best solution found

4.1. Heuristic LGFi

Note that LGFi is a two-stage heuristic. In the *preprocessing stage* items are sorted into a list, while in the *packing stage* these items are packed from the list into bins. More specifically, in the preprocessing stage items are sorted by non-increasing area as a first criterion. Ties are broken by non-increasing absolute difference between height and width of the items. The packing stage is an iterative process in which the following actions are performed at each iteration. First, the bottom leftmost position at which an item may be placed is identified. This position is henceforth called the *current position*. Then, two gaps are calculated with respect to this position. The *horizontal gap* is defined as the distance between the current position and either the right border of the bin or the left edge of the first item between the current position and the right border of the bin. The distance between the current position and the upper border of the bin defines the value of the *vertical gap*. The value of the smaller gap is called *current gap*. The current gap is compared to either the widths of the items from the list of unpacked items, if the horizontal gap is the current gap, or to the heights of the items from the list of unpacked items, if the vertical gap is the current gap. The first item that fills the gap completely is placed with its bottom left corner at the identified position. If no such item exists, the first item which fits without any overlap is placed with its bottom left corner at the current position. If no such item exists either, some of the area must be declared *wastage area*, which works as follows. A wastage area with the width of the horizontal gap is created. The height of the wastage area is chosen as the height of the upper edge of the lowest neighboring item, or, if no neighboring item exists, as the height of the bin. Finally, if no current position can be found, and if unpacked items exist, a new bin is opened.

4.2. The Evolutionary Algorithm

A solution in the context of the proposed algorithm—henceforth labelled EA-LGFi—is an input sequence s for LGFi. Note that any input sequence s is a permutation of all items that must be packed. The item at position j of this permutation (where $j = 1, \ldots, n$) is denoted by s_j. The objective function value $f(s)$ of a solution s is calculated by applying LGFi to s. The pseudo-code of EA-LGFi is shown in Alg. 1. The first step of EA-LGFi consists in generating the initial population of size p_{size} (see function GenerateInitialPopulation$(p_{\text{size}}, \kappa)$). Then, at each iteration a crossover operator is applied in function Crossover$(P, c_{\text{rate}}, \delta)$, generating $\lfloor c_{\text{rate}} \cdot p_{\text{size}} \rfloor$ new solutions, where $0 < c_{\text{rate}} \leq 1$. This results in a new population P' of less than p_{size} solutions. The missing $p_{\text{size}} - |P'|$ solutions are generated by function AddNewSolutions$(P', p_{\text{size}}, \kappa)$. In the following the three functions of algorithm EA-LGFi are outlined in more detail.

GenerateInitialPopulation(p_{size}, κ). In this function, p_{size} solutions are probabilistically generated as follows. First, let us denote the input sequence in which items are ordered with respect to non-increasing area as the *deterministic input sequence*. The position of an item i in the deterministic input sequence is henceforth called pos_i. Any solution s for the initial population is probabilistically generated based on the deterministic input sequence. This is done as follows. First, remember that the total number of items is denoted by n. A value v_i is then assigned to each item i in the following way:

$$v_i := (n - pos_i)^{\kappa} \tag{11}$$

where $\kappa \geq 1$ is one of the input parameters of the function. The positions of s are filled from 1 to n in an iterative way. At each step, let $\mathcal{I}' \subseteq \mathcal{I}$ be the set of items that are not yet assigned to s. An item $i \in \mathcal{I}'$ is chosen according

to probabilities $\mathbf{p}(i|\mathcal{I}')$ (for all $i \in \mathcal{I}'$) by roulette-wheel-selection. These probabilities $\mathbf{p}(i|\mathcal{I}')$ are calculated proportional to v_i:

$$\mathbf{p}(i|\mathcal{I}') = \frac{v_i}{\sum_{i \in \mathcal{I}'} v_i} \tag{12}$$

Note that the larger the value of parameter κ, the more similar the newly generated input sequence s will be to the deterministic input sequence.

Crossover(P, c_{rate}, δ). This operator, which is inspired by *uniform order-based crossover* [25], applies recombination to each of the best $\lfloor c_{\text{rate}} \cdot |P| \rfloor$ solutions of P, where $0 < c_{\text{rate}} \leq 1$ is a parameter of the algorithm. Extensive empirical tests have shown that a crossover rate $c_{\text{rate}} = 0.7$ works best for the instances at hand. For each solution s from the set of best $\lfloor c_{\text{rate}} \cdot |P| \rfloor$ solutions of P, a crossover partner $s^c \in P$ (such that $s^c \neq s$) is chosen from P by means of roulette-wheel-selection. Assume that P is an ordered list in which solutions are sorted according to their objective function values in a non-increasing manner. Ties are broken by the load of the last bin, that is, solutions with a lower load in the last bin are ordered first. Let $pos(s)$ denote the position of a solution s in P. The probability $\mathbf{p}(s^c|s)$ for a solution $s^c \neq s$ to be chosen as a crossover partner for solution $s \in P$ is as follows:

$$\mathbf{p}(s^c|s) := \frac{(p_{\text{size}} - 1 - pos(s^c))^\delta}{\sum_{s^o \in P, s^o \neq s} (p_{\text{size}} - 1 - pos(s^o))^\delta} \tag{13}$$

where $\delta \geq 1$ is a parameter of the algorithm. Given two crossover partners s and s^c, one offspring solution s^{off} is generated as explained in the following. First, three pointers (k, l and r) are initialized to the first position. Then, the n positions of s^{off} are filled from 1 to n as follows. If $s_k = s^c_l$ then $s^{\text{off}}_r := s_k$. In other words, if position k of solution s and position l of solution s^c contain the same item, then this item is placed at position r of the offspring solution s^{off}. Next, position pointer r is incremented, and position pointers k and l are moved to the right until reaching the closest position containing an item which does not yet appear in solution s^{off}. In case $s_k \neq s^c_l$, the item for position r of solution s^{off} is chosen probabilistically among s_k and s^c_l, where a probability of 0.75 is given to the item originating from the better of the two solutions. Afterwards, the position pointer r is incremented. Moreover, the position pointer of the solution from which the item was selected is moved to the right until reaching the closest position containing an item which does not yet appear in solution s^{off}. The resulting solution s^{off} is evaluated by using it as input for LGFi. In case $f(s^{\text{off}}) < f(s)$ or $f(s^{\text{off}}) = f(s)$ and s^{off} has a lower load than s in the last bin, solution s^{off} is added to the new population P', otherwise solution s is added to P'.

AddNewSolutions(P', p_{size}, κ). This function probabilistically generates $p_{\text{size}} - |P'|$ solutions in the same way as in function GenerateInitialPopulation$(p_{\text{size}}, \kappa)$.

5. Experimental Evaluation

EA-LGFi was implemented in ANSI C++ using GCC 4.4 for compiling the software. The experimental results that we outline in the following were obtained on a PC with an AMD64X2 4400 processor and 4 Gigabyte of memory. The proposed algorithm was applied to a benchmark set of 500 problem instances from the literature.[2] After an initial study of the algorithms' behavior, a detailed experimental evaluation is presented.

5.1. Problem Instances

Ten classes of problem instances for the 2BP are provided in the literature. A first instance set, containing six classes (I-VI), was proposed by Berkey and Wang in [15]. For each of these classes, the widths and heights of the items were chosen uniformly at random from the intervals presented in Table 1. Moreover, the classes differ in the width (W) and the height (H) of the bins. Instance sizes, in terms of the number of items, are taken from $\{20, 40, 60, 80, 100\}$. Berkey and Wang provided 10 instances for each combination of a class with an instance size. This results in a total of 300 problem instances.

[2] These 500 instances can be downloaded from `http://www.or.deis.unibo.it/research.html`.

Table 1. Specification of instance classes I-VI (as provided by [15]).

Class	w_j	h_j	W	H
I	[1,10]	[1,10]	10	10
II	[1,10]	[1,10]	30	30
III	[1,35]	[1,35]	40	40
IV	[1,35]	[1,35]	100	100
V	[1,100]	[1,100]	100	100
VI	[1,100]	[1,100]	300	300

The second instance set, consisting of classes VII-X, was introduced by Martello and Vigo in [26]. In general, they considered four different types of items, as presented in Table 2. The four item types differ in the limits for the width w_i and the height h_i of an item. Then, based on these four item types, Martello and Vigo introduced four classes of instances which differ in the percentage of items they contain from each type. As an example, let us consider an instance of class VII. 70% of the items of such an instance are of type 1, 10% of the items are of type 2, further 10% of the items are of type 3, and the remaining 10% of the items are of type 4. These percentages are given per class in Table 3. As in the case of the first instance set, instance sizes are taken from $\{20, 40, 60, 80, 100\}$. The instance set by Martello and Vigo consists of 10 instances for each combination of a class with an instance size. This results in a total of 200 problem instances.

Table 2. Item types for classes VII-X (as introduced in [26]). Note that for all classes it holds that $W = H = 100$.

Item type	w_j	h_j
1	$[\frac{2}{3} \cdot W, W]$	$[1, \frac{1}{2} \cdot H]$
2	$[1, \frac{1}{2} \cdot W]$	$[\frac{2}{3} \cdot H, H]$
3	$[\frac{1}{2} \cdot W, W]$	$[\frac{1}{2} \cdot H, H]$
4	$[1, \frac{1}{2} \cdot W]$	$[1, \frac{1}{2} \cdot H]$

Table 3. Specification of instance classes VII-X (as provided by [26]).

Class	Type 1	Type 2	Type 3	Type 4
VII	70%	10%	10%	10%
VIII	10%	70%	10%	10%
IX	10%	10%	70%	10%
X	10%	10%	10%	70%

5.2. Tuning Experiments

First of all we chose a fixed number of solution evaluations as stopping criterion for each run of EA-LGFi. In particular, after initial experiments we chose a budget of 10^6 solution evaluations for each run of EA-LGFi. Moreover, the value of parameter κ was set to 10 after initial experiments by hand.

The parameters that were selected for tuning are the population size p_{size} and δ. Remember that the value of δ is used for the calculation of the probabilities for solutions to be selected as crossover partners. More specifically, we considered $p_{\text{size}} \in \{10, 100\}$ and $\delta \in \{1, 5, 10, 15, 20\}$. In general, the higher the value of δ, the more are good solutions preferred over worse ones, when selecting a crossover partner s^c for a solution s. Then, EA-LGFi was applied exactly once for each combination of δ and p_{size}, to each of the 500 problem instances. The sum of the number of bins used in the best solutions generated for all 500 instances is shown in Figure 1 for each parameter value combination. Even though differences in algorithm performance are quite small, higher values of δ seem to work better than smaller ones. Moreover, a population size of 10 generally seems to work slightly better than a

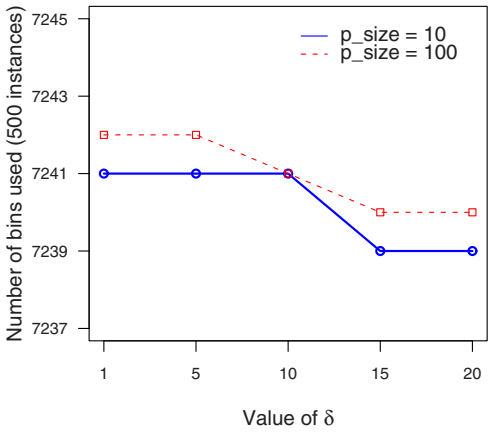

Fig. 1. Tuning results for EA-LGFi.

population size of 100. The final results of EA-LGFi presented in the following section are the ones obtained with $\delta = 20$ and $p_{\text{size}} = 10$.

5.3. Numerical Results

The results of EA-LGFi are compared in the following with the two best algorithms currently available for the 2BP problem: The set covering heuristic (SCH) from [16] and the hybrid GRASP approach from [22].

Numercial results are shown in Table 4 in a way which is traditional for the 2BP problem. Each table row presents the results for the 10 instances of a combination between instance class (I – X) and number of items (20 – 100). Concerning the columns, the one labelled **LB** contains the sum of the best lower bound values known from the literature for the 10 corresponding problem instances. The results of the compared algorithms are provided in two columns per algorithm. The first column (with heading **value**) provides the sum of the number of bins used in the best solutions generated for the 10 corresponding problem instances. For example, the best solutions generated for the 10 instances of Class I (20 items) by algorithm SCH occupy in total 71 bins. In case a value corresponds to the best performance obtained by any algorithm, it is highlighted in bold. Moreover, in the case of EA-LGFi a value is marked by an asterisk if it is better than the best know value as of today. The second column (with heading **time (s)**) shows the average computation time (in seconds) necessary to find the best solutions for the 10 problem instances of a combination between instance class and number of items. For example, algorithm SCH needed on average 0.06 seconds to find its best solutions for the 10 instances of Class I (20 items). Finally, the last line of Table 4 provides a summary of the results over all 500 problem instances. For each algorithm is given the sum of number of bins used, as well as the average computation time for the 500 instances.

There are several aspects about the results that should be mentioned. First, the number of bins used by the best solutions generated by EA-LGFi for the 500 problem instances amounts to 7239, which is the best value ever achieved by any algorithm. The best algorithm so far (GRASP) achieved a value of 7241. It is also interesting to note that EA-LGFi is able to solve four problem instances to optimality that have never been solved before. This concerns instances 173 and 174 (both from Class IV, with 60 items), instance 197 (from Class IV, with 100 items), and instance 298 (from Class VI, with 100 items).

Finally, we would like to comment on the computation times. Due to the fact that different processors and different computation time limits have been used for the generation of the results, the computation times are certainly not directly comparable. However, the computation times of all algorithms are, in general, very low. Therefore,

Table 4. Numerical results for the 500 instances of all 10 instance classes.

	LB	SCH		GRASP		EA-LGFi	
		value	time (s)	value	time (s)	value	time (s)
Class I							
20	71	**71**	0.06	**71**	0.00	**71**	0.00
40	134	**134**	2.42	**134**	0.00	**134**	0.00
60	197	**200**	7.26	**200**	4.50	**200**	0.01
80	274	**275**	4.63	**275**	1.50	**275**	0.00
100	317	**317**	5.21	**317**	0.00	**317**	0.00
Class II							
20	10	**10**	0.06	**10**	0.00	**10**	0.00
40	19	**19**	0.67	**19**	0.00	**19**	0.00
60	25	**25**	0.07	**25**	0.00	**25**	0.00
80	31	**31**	0.07	**31**	0.00	**31**	0.00
100	39	**39**	0.79	**39**	0.00	**39**	0.00
Class III							
20	51	**51**	0.07	**51**	0.00	**51**	0.02
40	92	**94**	2.66	**94**	3.00	**94**	0.01
60	136	**139**	6.21	**139**	4.60	**139**	0.27
80	187	**189**	8.80	**189**	4.10	**189**	20.68
100	221	**223**	12.80	**223**	4.90	224	26.17
Class IV							
20	10	**10**	0.06	**10**	0.00	**10**	0.00
40	19	**19**	0.07	**19**	0.00	**19**	0.00
60	23	25	6.15	25	3.00	**23***	12.18
80	30	32	10.35	**31**	1.90	**31**	0.00
100	37	38	4.72	38	1.50	**37***	0.00
Class V							
20	65	**65**	0.06	**65**	0.00	**65**	0.00
40	119	**119**	1.98	**119**	0.00	**119**	0.03
60	179	**180**	1.93	**180**	1.50	**180**	0.14
80	241	**247**	20.66	**247**	9.00	**247**	0.03
100	279	**282**	18.50	**282**	5.20	284	27.33
Class VI							
20	10	**10**	0.06	**10**	0.00	**10**	0.00
40	15	**17**	6.85	**17**	3.00	**17**	0.03
60	21	**21**	0.66	**21**	0.10	**21**	0.00
80	30	**30**	0.23	**30**	0.00	**30**	0.00
100	32	34	6.29	34	3.00	**32***	0.58
Class VII							
20	55	**55**	0.13	**55**	0.00	**55**	0.00
40	109	**111**	3.02	**111**	3.00	**111**	0.01
60	156	**158**	8.85	159	4.50	159	0.00
80	224	**232**	54.79	**232**	12.00	**232**	0.00
100	269	**271**	25.06	**271**	3.10	**271**	0.01
Class VIII							
20	58	**58**	0.06	**58**	0.00	**58**	0.03
40	112	**113**	0.96	**113**	1.50	**113**	0.00
60	159	162	9.05	**161**	4.20	**161**	0.02
80	223	**224**	11.60	**224**	1.60	**224**	0.00
100	274	279	47.13	278	6.10	**277**	0.25
Class IX							
20	143	**143**	0.06	**143**	0.00	**143**	0.00
40	278	**278**	0.07	**278**	0.00	**278**	0.00
60	437	**437**	0.07	**437**	0.10	**437**	0.00
80	577	**577**	0.08	**577**	0.00	**577**	0.00
100	695	**695**	0.11	**695**	0.00	**695**	0.00
Class X							
20	42	**42**	0.12	**42**	0.00	**42**	0.02
40	74	**74**	0.11	**74**	0.00	**74**	0.00
60	98	101	8.89	**100**	4.50	101	0.71
80	123	**128**	38.26	129	9.40	**128**	0.06
100	153	**159**	55.77	**159**	9.20	160	0.08
Summary	7173	7243	7.90	7241	2.21	**7239**	1.77

no algorithm can be identified to have a particular advantage or disadvantage over the other algorithms for what concerns the computation time requirements. In the following we provide the information about processors and computation time limits for the competitor algorithms: SCH was run on a Digital Alpha 533 MHz with a time limit of 100 seconds per instance. GRASP was executed on a Pentium Mobile with 1500 MHz with a stopping criterion of 50000 iterations per application.

6. Conclusions and Outlook

In this paper we have presented a rather simple hybrid evolutionary algorithm for tackling the oriented two-dimensional bin packing problem under free guillotine cutting (2BP). The proposed algorithm is strongly based on a probabilistic version of an existing one-pass heuristic (LGFi) from the literature. The results have shown that the proposed algorithm obtains very good results in comparison to current state-of-the-art approaches. In fact, four problem instances that were never solved to optimality before, could be solved to optimality. We can be sure of that because the objective function value of the best solution found is equal to the best lower bound value. In summary, the best solutions generated by the evolutionary algorithm for the 500 instances use 7239 bins in total. This is the best value ever achieved by any algorithm proposed for the 2BP.

In the future we plan to investigate additional ways in which the probabilistic version of LGFi might be exploited. For example, an ant colony optimization approach might be better suited than an evolutionary algorithm for learning input sequences for LGFi. Moreover, we plan to add a local search procedure to our algorithms for improving the constructed solutions.

Acknowledgements

This work was supported by grant TIN2012-37930-C02-02 of the Spanish Government. In addition, Christian Blum acknowledges support from IKERBASQUE, the Basque Foundation for Science.

References

[1] A. Lodi, S. Martello, D. Vigo, Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, INFORMS Journal on Computing 11 (4) (1999) 345–357.
[2] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.
[3] E. Hopper, B. Turton, A genetic algorithm for a 2D industrial packing problem, Computers and Industrial Engineering 37 (1–2) (1999) 375–378.
[4] P. E. Sweeney, E. R. Paternoster, Cutting and packing problems: A categorized, application-orientated research bibliography, The Journal of the Operational Research Society 43 (7) (1992) 691–706.
[5] C. Blum, J. Puchinger, G. Raidl, A. Roli, Hybrid metaheuristics in combinatorial optimization: A survey, Applied Soft Computing 11 (6) (2011) 4135–4151.
[6] L. Wong, L. S. Lee, Heuristic placement routines for two-dimensional bin packing problem, Journal of Mathematics and Statistics 5 (4) (2009) 334–341.
[7] A. Lodi, S. Martello, D. Vigo, Recent advances on two-dimensional bin packing problems, Discrete Applied Mathematics 123 (1-3) (2002) 379–396.
[8] A. Lodi, S. Martello, D. Vigo, Two-dimensional packing problems: A survey, European Journal of Operational Research 141 (2) (2002) 241–252.
[9] A. Lodi, Algorithms for two-dimensional bin packing and assignment problems, Ph.D. thesis, Università degli Studio di Bologna (1999).
[10] K. A. Dowsland, W. B. Dowsland, Packing problems, European Journal of Operational Research 56 (1) (1992) 2–14.
[11] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, R. E. Tarjan, Performance bounds for level-oriented two-dimensional packing algorithms, SIAM Journal on Computing 9 (4) (1980) 808–826.
[12] B. S. Baker, E. G. Coffman Jr., R. L. Rivest, Orthogonal packings in two dimensions, SIAM Journal on Computing 9 (4) (1980) 846–855.
[13] J. B. G. Frenk, G. Galambos, Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem, Computing 39 (3) (1987) 201–217.
[14] F. R. K. Chung, M. R. Garey, D. S. Johnson, On packing two-dimensional bins, SIAM Journal on Algebraic and Discrete Methods 3 (1) (1982) 66–76.
[15] J. O. Berkey, P. Y. Wang, Two dimensional finite bin packing algorithms, Journal of the Operational Research Society 38 (5) (1987) 423–429.
[16] M. Monaci, P. Toth, A set-covering-based heuristic approach for bin-packing problems, Informs Journal on Computing 18 (1) (2006) 71–85.

[17] T. G. Crainic, G. Perboli, R. Tadei, Extreme point-based heuristics for three-dimensional bin packing, Informs Journal on Computing 20 (3) (2008) 368–384.
[18] A. Lodi, S. Martello, D. Vigo, Approximation algorithms for the oriented two-dimensional bin packing problem, European Journal of Operational Research 112 (1) (1999) 158–166.
[19] O. Faroe, D. Pisinger, M. Zachariasen, Guided local search for the three-dimensional bin-packing problem, Informs Journal on Computing 15 (3) (2003) 267–283.
[20] K.-H. Loh, B. Golden, E. Wasil, A Weight Annealing Algorithm for Solving Two-dimensional Bin Packing Problems, Vol. 47 of Research/Computer Science Interfaces, Springer, New York, NY, 2009, pp. 121–146.
[21] M. A. Boschetti, A. Mingozzi, The two-dimensional finite bin packing problem. part II: New lower and upper bounds, 4OR 1 (2003) 135–147.
[22] F. Parreño, R. Alvarez-Valdes, J. F. Oliveira, J. M. Tamarit, A hybrid GRASP/VND algorithm for two- and three-dimensional bin packing, Annals of Operations Research 179 (1) (2010) 203–220.
[23] D. Pisinger, M. Sigurd, Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem, INFORMS Journal on Computing 19 (1) (2007) 36–51.
[24] J. Puchinger, G. Raidl, Models and algorithms for three-stage two-dimensional bin packing, European Journal of Operational Research 183 (3) (2007) 1304–1327.
[25] L. Davis, Chapter 6: Order-Based Genetic Algorithms and the Graph Coloring Problem, Van Nostrand Reinhold, New York, NY, 1991, pp. 72–90.
[26] S. Martello, D. Vigo, Exact solution of the two-dimensional finite bin packing problem, Management Science 44 (3) (1998) 388–399.