Computers & Operations Research 40 (2013) 991–1003

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

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

Heuristics for the strip packing problem with unloading constraints ☆

Jefferson L.M. da Silveira *, Flávio K. Miyazawa, Eduardo C. Xavier

Institute of Computing, University of Campinas – UNICAMP, Campinas, SP, Brazil

ARTICLE INFO

Available online 22 November 2012

Keywords:
Strip Packing
GRASP
Approximation algorithms

ABSTRACT

This article addresses the Strip Packing Problem with Unloading Constraints (SPU). In this problem, we are given a strip of fixed width and unbounded height, and n items of C different classes. As in the well-known two-dimensional Strip Packing problem, we have to pack all items minimizing the used height, but now we have the additional constraint that items of higher classes cannot block the way out of lower classes items. This problem appears as a sub-problem in the *Two-Dimensional Loading Capacitated Vehicle Routing Problem* (2L-CVRP), where one has to optimize the delivery of goods, demanded by a set of clients, that are transported by a fleet of vehicles of limited capacity based at a central depot. We propose two approximation algorithms and a GRASP heuristic for the SPU problem and provide an extensive computational experiment with these algorithms using well know instances for the 2L-CVRP problem as well as new instances adapted from the Strip Packing problem.

1. Introduction

In recent years some attention has been devoted to the combination of two problems: the two-dimensional packing and the routing problem. The combination of these two problems models situations where one aims to deliver goods, demanded by customers, that are transported by vehicles of limited capacity based at a central depot. This problem is called *Two-Dimensional Loading Capacitated Vehicle Routing Problem* (2L-CVRP) [25]. The objective is to generate a set of routes of minimum total cost that covers all clients, where each route induces feasible packings, i.e., all items of one route must be packed in one vehicle satisfying the traditional packing constraints and a new unloading constraint. The unloading constraint is the following: given a set of items that are delivered along a route, while delivering items of one client, there must not exist items of other clients ahead on the route blocking the way out of the items of the current client.

One important task in algorithms for the 2L-CVRP problem (see [20,18,26,15]) is to check if a giving route induces a valid packing. One way of doing this is solving a strip packing problem with the unloading constraint and checking whether the generated packing height is smaller or larger than the maximum allowable height. In this work we focused on this problem called here by Strip Packing with Unloading Constraints (SPU).

We can define the SPU problem as follows: given a strip S of fixed width and unbounded height, and a list of items of C different classes, each item a_i of height $h(a_i)$, width $w(a_i)$ and class $c(a_i)$, we must pack the items into S minimizing the used height. Furthermore, if an item a_i has class greater than a_j, i.e. $c(a_i) > c(a_j)$, then a_i must not block the way out when removing item a_j. We also consider the case in which 90° rotations are allowed (SPU^r). This problem is strongly NP-Hard since it is a generalization of the Two-dimensional Strip Packing problem.

Papers which addresses the 2L-CVRP problem used some simple heuristics or exact algorithms to tackle the packing problem, and do not provide information about the quality of the solutions (except [18] that presented the average occupied area in the vehicles (bins) of the problem).

In [20] Iori et al. proposed an exact algorithm to the 2L-CVRP problem. Their packing algorithm is the bottom-left heuristic and a branch-and-bound procedure to check the feasibility of the loadings. Their solution can solve instances involving up to 25 clients and 91 items in 1 day of CPU time.

Gendreau et al. [18] proposed a tabu search algorithm to the 2L-CVRP problem. The packing problem is solved using heuristics, local search and a truncated branch-and-bound. Their algorithm iteratively applies a procedure based on the Touching Perimeter algorithm [27] for the two-dimensional bin packing problem (it is worth noting that the Touching Perimeter heuristic is also used in the two-dimensional strip packing problem [23]). At first, the items are sorted in reverse order of clients visit, and a packing is constructed. Subsequently the algorithm tries to improve the packing perturbing the trivial order that items were packed.

In [26] Kiranoudis et al. proposed a guided tabu search heuristic to the 2L-CVRP. They used five different heuristics (in

* Corresponding author.
E-mail addresses: jmoises@ic.unicamp.br (J.L.M. da Silveira), fkm@ic.unicamp.br (F.K. Miyazawa), eduardo@ic.unicamp.br (E.C. Xavier).

☆ This research was supported by CNPQ and FAPESP.

0305-0548/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2012.11.003

order) to tackle the packing problem. The first and second heuristics are based on the bottom-left heuristic. The third and fourth heuristics are similar to the one used by Gendreau et al. [18], based on the Touching Perimeter.

The best result for the 2L-CVRP is due to Doerner et al. [15] and their *Ant Colony Optimization* heuristic. The authors use *Bin-packing* lower bounds to prove unfeasibility of some routes, and then heuristics to construct the packing solutions. Their heuristics are quite similar to the ones previously cited in [18,26] (Bottom-left and Touching Perimeter) and also use a truncated branch-and-bound with a limited CPU time.

One of the best results for the Strip Packing problem without rotations was obtained by a Reactive GRASP heuristic proposed by Alvarez et al. [1]. Some other papers also achieved similar results [5,11,8].

In [3], Azar and Epstein proposed an online 4-competitive algorithm to a version of the strip packing problem, where while packing one item there must be a free way from the top of the bin until the position where the item is packed. In this model, a rectangle arrives from the top of S, and it should be moved continuously using only the free space until it reaches its place, as in the well known *TETRIS game*. Their online algorithm can be easily modified to an *offline* algorithm to the relaxed version of the SPU problem where the items can use vertical and horizontal movements to leave the bin.

Fekete et al. [16], proposed an online 2.6154-competitive algorithm to a version of the square strip packing problem, similar to the one considered in [3]. In this algorithm, the items are packed from the top of S and are moved only with vertical movements to reach its final position. In addition, an item is not allowed to move upwards and has to be supported from below when reaching its final position. These conditions are called *gravity constraints*. Their slot based algorithm can be easily used to the SPU problem, achieving an 2.6154-approximation, in the special case where items are squares. We just need to sort the items in non-increasing order of class values.

Finally, Augustine et al. [2] present approximation algorithms forr a related problem. They consider the strip packing problem with precedence constraints and/or with release dates. Their problem has applications in scheduling problems for FPGA.

1.1. Our results

For the SPU^r problem, we propose a bin packing based heuristic and prove that this heuristic is a 6.75-approximation algorithm. Besides that, we also propose an 1.75-approximation algorithm for a special case of the SPU problem, where the number of classes (clients in a route) is bounded by a constant. This algorithm is based on the well known *First-Fit-Decreasing Height* algorithm [14].

Finally, we propose a GRASP heuristic for the SPU problem that is based on the Reactive GRASP heuristic presented in [1]. We adapted this heuristic to consider the unloading constraint and also for the SPU^r problem. We changed the focus of the algorithm to the items classes instead of their dimensions.

Besides the theoretical results presented for the approximation algorithms, their practical performance is also checked. The effectiveness of the proposed heuristics is demonstrated through extensive computational experiments on benchmark instances [30]. We also generated several new instances based on benchmark instances for the strip packing problem [31,4,6,9,12,21,22,7].

We show that our algorithms achieve a good occupation of the area of the strip in low CPU time. We also show that our best packing heuristics improves the solutions of the 2L-CVRP problem when compared to other well know heuristic.

1.2. Paper organization

This paper is organized as follows: in Section 2 we introduce our definitions and formalize the description of the SPU problem. The approximation algorithms are presented in Section 3. In Section 3.1 we present an asymptotic 6.75-approximation algorithm for the SPU^r problem and in Section 3.2 we present an asymptotic 1.75-approximation algorithm for the special case of the SPU problem, where the number of classes in an instance is bounded by a constant. In Section 4 we present the constructive algorithm and the Local Search strategy used in the GRASP based heuristics which are described in Section 5. In Section 6 we present the instances used on the experiments. In Section 7 we summarize our computational experiments and results. Moreover, we present lower bounds used in this work. Finally, in Section 8 we analyze the results and argue about the effectiveness of the proposed heuristics and approximation algorithms.

2. Definitions and notation

We define the SPU problem as follows: an instance of the problem is composed by a strip S of fixed width W and unbounded height, and a list L of n items, each item a_i with height $h(a_i)$, width $w(a_i)$ and class $c(a_i)$. The class values $c(a_i)$ are interpreted as an order of removal of the item from the strip. A packing is defined by a function that maps each item a_i to a point $(x(a_i),y(a_i))$, where $x(a_i)$ and $y(a_i)$ are the coordinates of the bottom-left corner of the item a_i on S. The bottom-left corner of the strip has coordinates (0,0). The goal is to pack all items into S minimizing $\max_i\{y(a_i)+h(a_i)\}$, $1 \le i \le n$, subject to the constraints:

- All the items must be completely contained in S.
- Items can not overlap each other.
- All the items must satisfy the unloading constraint (see Fig. 1), i.e., for any two items $a_i,a_j \in L$, where $c(a_i) > c(a_j)$, we must have $x(a_i)+w(a_i) \le x(a_j)$ or $x(a_j)+w(a_j) \le x(a_i)$ or $y(a_i)+h(a_i) \le y(a_j)$. This imposes that each item can be removed from the strip in increasing order of classes using only vertical movements.

Let $\mathcal{A}$ be an algorithm for the SPU problem and let $\mathcal{A}(I)$ be the cost of the solution computed by $\mathcal{A}$ for instance I. We say that $\mathcal{A}$ is an α-approximation algorithm if it has polynomial time complexity, and for every I it satisfies $\mathcal{A}(I) \le \alpha\mathrm{OPT}(I)$, where $\mathrm{OPT}(I)$ is the cost of an optimum solution to instance I. As it is common in packing problems, we consider in this work asymptotic approximation algorithms, where in this case the algorithm must satisfy $\mathcal{A}(I) \le \alpha\mathrm{OPT}(I)+\beta$ for some constant β (Fig. 1).

3. Approximation algorithms

3.1. A 6.75-approximation algorithm for the SPU^r problem

In this section we present the *Hybrid Bin Packing* (HBP) algorithm to solve the SPU^r problem. Without loss of generality, we assume that the width of the strip is 1 and all items have width and height at most 1. The HPB algorithm computes the solution in two stages. The algorithm uses in the first stage, a bin packing algorithm, which we call *Level Bin Packing* (LBP). This bin packing algorithm packs the items into bins (rectangles) of height 1 and width 1, using levels or, what we call, horizontal sub-strips (horizontal slices of a bin, see Fig. 2). Then the HBP algorithm, using the bins computed by the LBP algorithm, concatenates these bins in such a way to obtain a feasible strip S. Due to the form that

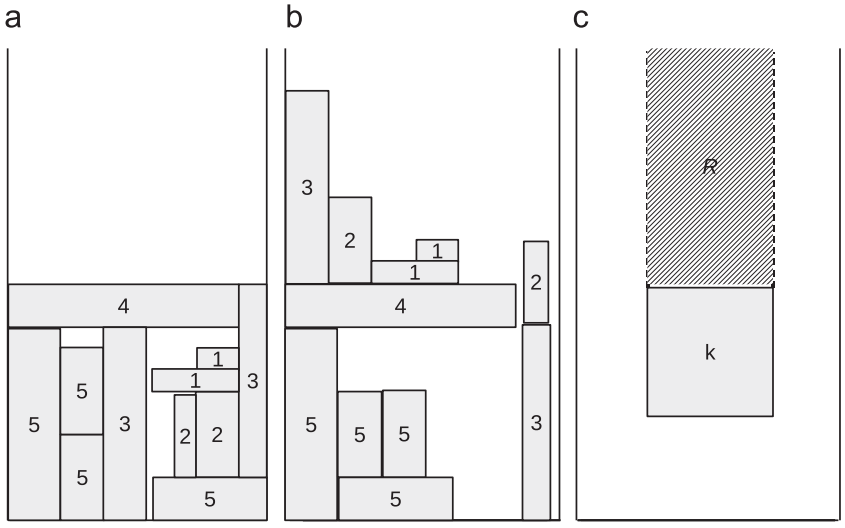

Fig. 1. For each item in the figure, the number on it corresponds to $c(a_i)$ (the order of removal). In part (a) we can see an infeasible packing since an item of class 4 is blocking the way out of items of classes 1, 2 and 3. In part (b) we have a feasible packing. In part (c) we see a forbidden area R to items a_i with $c(a_i) > k$ due to the unloading constraint.

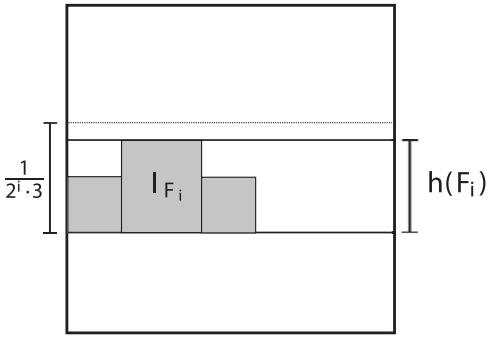

Fig. 2. In this figure we represent the definitions for the HBP algorithm. Non-buffer items are packed in sub-strips F_i from left to right. I_{F_i} is the height of the tallest item in the sub-strip. In this example, $S(B_k)$ contains only one sub-strip.

the algorithm HBP concatenates the bins, we can resize their widths to values smaller than 1. We first present the LBP algorithm (see Algorithm 1), and then we present the HBP algorithm (see Algorithm 2).

To describe the LBP algorithm we need some definitions. The items are divided in two categories: *buffers* and *non-buffers*. Buffers are items where at least one dimension is larger than 1/3 and non-buffers the remaining items.

The LBP algorithm creates several bins, and we denote by B_k the kth bin used by LBP. The occupied height of some bin B_k is denoted by $h(B_k)$. Formally $h(B_k) = \max_i\{h(a_i)+y(a_i)\}, a_i \in B_k$, i.e, it is the maximum height where there is some item a_i packed. Similarly, we denote by $w(B_k)$ the occupied width of bin B_k. Formally $w(B_k) = \max_i\{w(a_i)+x(a_i)\}, a_i \in B_k$.

A *buffer* item is packed alone in a sub-strip with height equal to its height or in individual bins. A sub-strip used to pack a *buffer* is considered *full*.

Non-buffer items are packed into sub-strips F_j of height $h(F_j) = 1/(3 \cdot 2^j)$, for $j = 0,1,\ldots$. A *non-buffer* item a_i is packed in a sub-strip F_j, such that $1/(3 \cdot 2^{j+1}) < h(a_i) \le 1/(3 \cdot 2^j) = h(F_j)$. We say that a_i has *type* F_j (see Fig. 2). Notice that the type of some item a_i is uniquely defined by its height $h(a_i)$.

The non-buffer items are packed into sub-strips from left to right side by side, justified at the bottom of the sub-strip. We consider a sub-strip F_j *full* if its occupied width satisfies $w(F_j) \ge \frac{2}{3}$.

For each sub-strip F_j, we denote by T_{F_j} the tallest item packed in F_j. Sub-strips keep their original height until the time they are closed when we set $h(F_j) = h(T_{F_j})$. When a sub-strip is closed, no more items can be packed on it. When a bin or a sub-strip is opened, items can be packed on them.

Finally we denote by $S(B_k)$ the set of all sub-strips in bin B_k (see Fig. 2).

The LBP algorithm (see Algorithm 1) always keep at most one open bin and at most one open sub-strip of each height $h(F_j) = 1/(3 \cdot 2^j)$, for $j = 0,1,\ldots$. A sub-strip is closed when it becomes full or when the bin containing it is closed.

The LBP algorithm packs items in non-increasing order of values $c(a_i)$ (line 5). For each class, it first packs *buffers* $b_0,\ldots,b_x$ into the last used bin in separated sub-strips while they fit (line 6). Then it packs each remaining *buffer* rotated, such that $w(b_i) \le h(b_i)$, alone in a bin of width $w(b_i)$ (the bin is closed after packing the item (line 7)).

After packing *buffers* of one class, the algorithm packs the *non-buffer* items (lines 8–11). For each *non-buffer* item a_i, the algorithm packs it into an open sub-strip F_j of height $1/(3 \cdot 2^j)$ such that $1/(3 \cdot 2^{j+1}) < h(a_i) \le 1/(3 \cdot 2^j)$ (line 9). If the item cannot be packed in F_j, then this sub-strip must be *full*, and then it is closed; a new sub-strip is created and a_i is packed there. If after packing a *non-buffer* a_i we have $\sum_{F \in S(B_k)} T_F > 1$ then the algorithm removes a_i, and packs it into a new sub-strip in a new bin and closes the last bin (line 11).

Algorithm 1. Level Bin Packing (LBP).

1: **Input**: List L of items partitioned into C different classes.
2: **Begin**
3: Sort L by non-increasing order of class.
4: Rotate all items $a \in L$, such that $h(a) \le w(a)$.
5: **for** ($c = C$ down to 1) **do**
6: Pack the buffers of class c into the open Bin B_k while $\sum_{F \in S(B_k)} T_F \le 1$, each *buffer* in a new sub-strip.
7: Rotate each remaining buffer b_i, such that $w(b_i) \le h(b_i)$, and pack each one in a new bin of width $w(b_i)$. Close each one of these new bins.
8: **for** (each *non-buffer* a_i of class c) **do**
9: Pack a_i in the open sub-strip F_j where $1/(3 \cdot 2^{j+1}) < h(a_i) \le 1/(3 \cdot 2^j)$. If there is no such sub-strip, or a_i does

not fit in F_j, create a new sub-strip and pack a_i there. Close F_j if it exists and set $h(F_j) = T_{F_j}$.
10: **if** $(\sum_{F \in S(B_k)} T_F > 1)$ **then**
11: Remove item a_i from the current bin B_k and pack it into a new sub-strip into a new bin B_{k+1} at the bottom and close B_k.
12: **end if**
13: **end for**
14: **end for**
15: Return the created bins.
16: **end**.

The algorithm HBP, that generates the final solution, is presented in Algorithm 2. It just calls the algorithm LBP, concatenate all bins returned side by side forming a horizontal strip S of height 1 and width equal to the sum of the bins widths (containing all the bins). Then S is rotated to provide a solution to the original problem.

Algorithm 2. Hybrid Bin Packing (HBP).

1: **Input**: $L = \{a_1, a_2, \ldots, a_n\}$
2: **Begin**
3: Let $B_1, B_2, \ldots, B_m$ be the bins computed by LBP in the order they were created.
4: Concatenate $B_1, B_2, \ldots, B_m$ forming one strip S of height 1 and width $\sum_{k=1}^{m} w(B_k)$.
5: **Return** S rotated such that its width is 1 and its height is $\sum_{k=1}^{m} w(B_k)$.
6: **end**.

Theorem 1. *The packing produced by HBP satisfies the restrictions of the SPU problem.*

Proof. Let $B_1, \ldots, B_m$ be the bins created by LBP in the order they were created. These bins are concatenated in the order they were created, forming a strip S . For each successive pair of bins B_k and B_{k+1}, we guarantee that all items in B_{k+1} have class smaller than or equal to the items in B_k, since the algorithm packs items in non-increasing order of classes. So items in one bin will not block items of previous bins. Inside each bin, the feasibility of the solution is also guaranteed by the packing in non-increasing order of classes, since each item is packed in a sub-strip in the leftmost position to the right of previous packed items. Also notice that different sub-strips do not interfere with each other, since all items are packed completely inside a sub-strip. □

3.1.1. HBP analysis

In this section we prove that the HBP is a 6.75-approximation algorithm. First we present some results that guarantee a fraction of occupied area by items on each bin created by the algorithm.

Lemma 2. *The sub-strips opened to pack buffers and the full sub-strips are at least $\frac{1}{3}$ full.*

Proof. In the first case, where a sub-strip contains a buffer b_i, the height of the sub-strip is $h(b_i)$ and since $w(b_i) \geq 1/3$ we have an occupation of at least $(h(b_i) \cdot w(b_i))/h(b_i) \geq \frac{1}{3}$ of the sub-strip surface.

In a *full* sub-strip F_j, used to pack non-buffers, each packed item has height at least $1/(3 \cdot 2^{j+1})$. Since the sub-strip is full, its occupied width is at least $(1-1/3)$, then we can guarantee an occupation of

$$\frac{(\frac{1}{2} \cdot h(F_j)) \cdot \frac{2}{3}}{h(F_j)} \geq \frac{1}{3}$$

of the sub-strip surface. □

Lemma 3. *A bin contains at most $\frac{2}{3}$ of height used by non-full sub-strips.*

Proof. There is at most one *non-full* sub-strip of each type F_j (with height $1/(3 \cdot 2^j)$) in a bin. The total height of non-full sub-strips is bounded by $\sum_{j=0}^{\infty}(1/(3 \cdot 2^j)) = \frac{2}{3}$. □

Lemma 4.

(i) *Suppose that a fraction of height $1/3$ of a bin is used by full sub-strips and one non-full sub-strip F_0. The minimum area of packed items occurs when F_0 contains one item a_i with height $1/6$. The area of items in these sub-strips is at least $1/12$.*
(ii) *Suppose that a fraction of height $1/3$ of a bin is used by full and non-full sub-strips F_j with $j \geq 1$. The minimum area of packed items occurs when there is only non-full sub-strips F_j, for $j \geq 1$. The area of the packed items is at least $1/27$.*

Proof. First notice that by Lemma 2, a full sub-strip is at least $1/3$ full. The total area of items considering all these sub-strips is then at least $1/3$ times the height used by these sub-strips. Also notice that for a *non-full* sub-strip a worst case of occupation occurs when only one item is packed on it.

For (i), the minimum total area of items occurs when F_0 has only one item a_i. The area of the items in this fraction of the bin is

$$h(a_i)^2 + (1/3 - h(a_i)) \cdot 1/3 \quad (1)$$

since the area of a_i is at least $h(a_i)^2$ because $h(a_i) \leq w(a_i)$, and the remaining height $(1/3 - h(a_i))$ is at least $1/3$ full. The minimum of the function occurs when $h(a_i) = 1/6$. The minimum area of items is then at least $1/12$.

For (ii) suppose a fraction of height $1/(3 \cdot 2^j)$ that can be occupied by a *non-full* sub-strip F_j for $j \geq 1$, with one item a_i, or with *full* sub-strips. The equation of the total area of items in this fraction of height is similar to Eq. (1), except that the height $1/3$ is replaced by $1/(3 \cdot 2^j)$. The function is decreasing for $[0,1/6]$. In this case, where $j \geq 1$, the item a_i has height at most $1/(3 \cdot 2^j) \leq 1/6$. In order to minimize the function of total area of items, we have to set $h(a_i) = 1/(3 \cdot 2^j)$. So the minimum occupied area occurs when we use only *non-full* sub-strips, each one with maximum height. The total area of items in this case is at least $\sum_{j=1}^{\infty}(\frac{1}{3 \cdot 2^j})^2 = \frac{1}{27}$. □

Lemma 5. *The bins created by the LBP algorithm are full by at least $\frac{4}{27}$ on average, excluding perhaps the last bin.*

Proof. We will prove this Lemma by induction on the number of created bins, denoted by k.

When $k=1$ we have only one bin and the proof is trivial. Now assume that except for the last bin (B_{k-1}), the bins are at least $\frac{4}{27}$ full. Now consider the way in which bin B_k was opened. We will divide this step in two cases.

*Case*1: Consider that B_k was opened by a non-buffer item a_i because when packing it in B_{k-1}, we had $\sum_{F \in S(B_{k-1})} I_F > 1$. At least $1-h(a_i)$ of height is used by *full* (*F*) and *non-full* (*NF*) sub-strips in B_{k-1}. Suppose a_i has type F_j for some j. We can assume that this sub-strip F_j is not part of *NF*: when packing a_i in a *non-full* sub-strip F_j we must had $\sum_{F_l \in S(B_{k-1})} I_{F_l} > 1$, so the height $1-h(a_i)$ already exclude the height used by F_j. So in the height $1-h(a_i)$ used to pack sub-strips, we can assume that F_j is not in *NF*.

Suppose a_i has type F_0. In this case at least $(1-2/3)$ of height is used by *full* sub-strips, since the maximum height of a_i is $1/3$ and the maximum height of sub-strips in *NF* is also $1/3$. According to Lemma 4 the area of items in *NF* is at least $1/27$. So the total area of items in bin B_{k-1} is at least $(1-2/3)\cdot 1/3+1/27=4/27$.

Suppose a_i has *type* F_j, for $j\geq 1$. In this case at least $(1-1/6-2/3)$ of height is used exclusively by *full* sub-strips, since the maximum height of a_i is $1/6$ and the maximum height in *NF* is $2/3$. In these last $2/3$ of height, by Lemma 4 a minimum area of packed items occurs when in a fraction of $1/3$ of height is packed a sub-strip F_0 and *full* sub-strips (with total items area at least $1/12$), while in the other fraction of $1/3$ of height is packed only non-full sub-strips (with total items area $1/27$). The total area in bin B_{k-1} is at least $(1-1/6-2/3)\cdot 1/3+1/12+1/27>4/27$.

*Case*2: Consider that B_k was opened by a *buffer* b such that $w(B_k)=w(b)\leq h(b)$. First consider that $w(b)\geq 2/3$. In this case these bins B_{k-1} and B_k, are full by at least

$$\frac{h(b)\cdot w(b)}{1+w(b)}\geq \frac{w(b)^2}{1+w(b)}>\frac{4}{27},$$

since $w(b)\geq 2/3$.

Consider now that $2/3\geq w(b)\geq 0$. In this case there is at least $1/3$ of height in B_{k-1} used by *full* and *non-full* sub-strips. By Lemma 4 a worst case occurs when $1/3$ of the available height is occupied only by non *full* sub-strips F_j for $j\geq 1$ with total items area $1/27$. The possible remaining height $(1-1/3-w(b))$ must be used by *full* sub-strips of height h' and a *non-full* sub-strip F_0 with only one item a. The bins are full by at least

$$\frac{w(b)^2+1/27+h(a)^2+\frac{1}{3}h'}{1+w(b)},$$

where $1/6<h(a)\leq 1/3$, $h(a)+h'=1-w(b)-1/3$ and $2/3\geq w(b)>0$. The minimum of this function is $0.168517>\frac{4}{27}$. □

Theorem 6. *Let L be a list of rectangles, then* $\mathrm{HBP}(L)\leq 6.75\,\mathrm{OPT}(L)+1$.

Proof. According to Lemma 5, the average occupied area in S is at least $\frac{4}{27}$, except for the last bin created. So $(\mathrm{HBP}(L)-1)\frac{4}{27}\leq \sum_{a_i\in L} w(a_i)\cdot b(a_i)$ and then

$$\mathrm{HBP}(L)\leq 6.75\sum_{a_i\in L} w(a_i)\cdot b(a_i)+1\leq 6.75\mathrm{OPT}(L)+1. \qquad \square$$

3.2. An 1.75-approximation for the for SPU and SPU^r problems with a bounded number of classes

In this section we describe an 1.75-approximation algorithm called *First-Fit Decreasing Height by Class* (FFDHC), which uses the well-known *First-Fit Decreasing Height* (FFDH) algorithm [14] for the Strip Packing Problem. The FFDH is a level based algorithm that works as follows: first it sorts the items by non-increasing order of height and then, for each item a in this order, it packs a in the lowest sub-strip where it can be packed. If there is no such sub-strip, a new sub-strip is created with height $h(a)$, above all previous sub-strips, and a is packed on it.

The algorithm FFDHC (see Algorithm 3) works as follows: first the input list L is partitioned by class values into C different subsets. Then for each class $c\in C$, it packs items of this class using the FFDH algorithm but in a slightly different way: it packs first the items a_i with $w(a_i)>1/2$ and then the remaining items of class c. After that, it closes all sub-strips such that the following classes are packed in new sub-strips above the created ones.

Denote by $\mathrm{FFDH}(L)$ the packing generated by the algorithm FFDH over the list L.

Algorithm 3. First-Fit Decreasing Height by Class (FFDHC).

1: **Input**: A list L of items of C classes.
2: **Begin**
3: Let L_i be the list of items of class $i=1,\ldots,C$.
4: $L_i'=\{a: w(a)>1/2 \text{ and } a_i\in L_i\}$
5: $L_i''=\{a: w(a)\leq 1/2 \text{ and } a_i\in L_i\}$
6: Let $P=\emptyset$ be the final packing.
7: **for** $i=C$ down to 1 **do**
8: $P_i'=\mathrm{FFDH}(L_i')$
9: $P_i''=\mathrm{FFDH}(L_i'')$
10: Pack P_i' above all the previous sub-strips in P.
11: Pack P_i'' above all the previous sub-strips in P.
12: **end for**
13: **End**

The algorithm FFDHC clearly satisfies the unloading constraint: Items of each class are packed separated above previously packed classes, and classes are considered in non-increasing order.

3.2.1. FFDHC analysis

First we present a result about the FFDH algorithm (see [14]). Let $\mathrm{Area}(L)=\sum_{a_i\in L} w(a_i)\cdot h(a_i)$.

Lemma 7 (*Coffman et al. [14]*). *Let L be a list of rectangular items with area* $\mathrm{Area}(L)$, *where each item has width of at most* $1/m$, *for any integer* $m\geq 2$, *then*

$$\mathrm{FFDH}(L)\leq \frac{m+1}{m}\mathrm{Area}(L)+1.$$

Denote by $h(P_i')$ (resp. $h(P_i'')$) the sum of the height of the sub-strips packed in P_i' (resp. P_i''). Let L' be the list with items with width larger than $1/2$, i.e., $L'=\bigcup_{i=1}^{C} L_i'$, and let L'' be the remaining items, i.e., $L''=\bigcup_{i=1}^{C} L_i''$.

Theorem 8. *Let L be a list o rectangles of C different classes, then FFDHC is an*1.75 *asymptotic approximation algorithm for the SPU problem when C is bounded by a constant.*

Proof. Let $h'=\sum_{i=1}^{C} h(P_i')$ and $h''=\sum_{i=1}^{C}(h(P_i'')-1)$. We can conclude that $\mathrm{FFDHC}\ (L)\leq h'+h''+C$.

First notice that $\mathrm{OPT}(L)\geq h'$, since all items $a\in L'$ have $w(a)>1/2$ and then must be packed one above the other. Also notice that if $h''\leq 0$, then $\mathrm{FFDHC}\ (L)\leq h'+h''+C\leq h'+C\leq\ \mathrm{OPT}(L)+C$ and then the theorem is proved. So we assume that $h''>0$.

Since all items $a\in L_i''$ have $w(a)\leq 1/2$, then by Lemma 7, we have

$$\mathrm{Area}(L_i'')\geq \frac{2}{3}(\mathrm{FFDH}(L_i'')-1).$$

Then

$$\mathrm{Area}(L'')=\sum_{i=1}^{C}(\mathrm{Area}(L_i''))\geq \sum_{i=1}^{C}\frac{2}{3}(\mathrm{FFDH}(L_i'')-1)=\frac{2}{3}h''.$$

We also have that $\mathrm{OPT}(L)\geq \mathrm{Area}(L)$ and then

$$\mathrm{OPT}(L)\geq \mathrm{Area}(L)=\mathrm{Area}(L')+\mathrm{Area}(L'')\geq \frac{1}{2}h'+\frac{2}{3}h''.$$

Then, we have

$$\mathrm{OPT}(L) \geq \max\left\{h', \left(\frac{1}{2}h' + \frac{2}{3}h''\right)\right\}.$$

So,

$$\begin{aligned}\mathrm{FFDHC}(L) &\leq h' + h'' + C \leq (h' + h'')\frac{\mathrm{OPT}(L)}{\max\left\{h', \frac{1}{2}h' + \frac{2}{3}h''\right\}} + C \\ &= \alpha\,\mathrm{OPT}(L) + C,\end{aligned}$$

where $\alpha = (h' + h'')/\max\{h', \frac{1}{2}h' + \frac{2}{3}h''\}$. Finally we have that $\alpha \leq 1.75$ by Miyazawa and Wakabayashi [28]. □

To consider the case in which rotations are allowed we only need to perform a new partition of L into L' and L'' such that: $L_i' = \{a : w(a) > 1/2, h(a) > 1/2, a_i \in L_i\}$ and $L_i'' = L_i \backslash L_i'$, where in L_i'' all items have width at most $1/2$ (rotations are performed if necessary to satisfy this). Then we use the algorithm FFDHC with these new list partitions. Denote this modified algorithm by FFDHC^r.

Using this partition all the arguments used in the Theorem 8 are valid and then the following result holds.

Theorem 9. *Let L be a list o rectangles of C different classes, then FFDHC^r is a* 1.75 *asymptotic approximation algorithm for the* SPU^r *problem when C is bounded by a constant.*

Proof. Similar to the proof of Theorem 8. □

Now we improve the approximation ratio of the FFDHC algorithm considering that the items are squares.

We are going to use the following result also from [14].

Lemma 10. *Let L be a list of square items of total area* Area(L), *then*

$$\mathrm{FFDH}(L) \leq \frac{3}{2}\mathrm{Area}(L) + 1.$$

We just need to do one simple modification on algorithm FFDHC: we set $L_i' = L_i$ and $L_i'' = \emptyset$ for $i = 1, \ldots, C$. Denote this algorithm by FFDHC ^{s} (*squares*). We have the following result.

Corollary 11. *Let L be a list of squares of C different classes. Then* $\mathrm{FFDHC}^s(L) \leq \frac{3}{2}\,\mathrm{OPT}(L) + C$.

Proof. This is a simple proof, based on Lemma 10

$$\begin{aligned}\mathrm{FFDHC}^s(L) &= \mathrm{FFDH}(L_C') + \cdots + \mathrm{FFDH}(L_1') \\ &\leq \sum_{i=1}^{C}\left(\frac{3}{2}\,\mathrm{Area}(L_i') + 1\right) \\ &= \frac{3}{2}\,\mathrm{Area}(L) + C \\ &\leq \frac{3}{2}\,\mathrm{OPT}(L) + C. \qquad \square\end{aligned}$$

3.3. Improving the FFDHC algorithm

In practical situations, the FFDHC algorithm can return poor solutions. For instance consider an example with only one item per class. In this case the algorithm just creates a pile with all items left justified on the strip. So we did two improvements on the FFDHC algorithm.

Notice that when we pack the items of class c they do not use strips that were previously opened for other classes, since this can brake the unloading constraints. So we close all opened sub-strips before packing the items of class c (see Fig. 3 part a). But notice that we can keep the top sub-strip used for a previous class opened, since items of the current class packed in this top sub-strip will not be blocked by items of previous packed classes. The first improvement is then to keep opened this last sub-strip of the last class packed (see Fig. 3 part b).

The second improvement deals with the empty space between items of subsequent sub-strips. Let S_i, $i = 1, \ldots, k$ be the sub-strips generated by the algorithm FFDHC for some list L of items. First we reverse the order of items packed on the even sub-strips (S_i where $i \equiv 0 \pmod 2$) and push them to the right most position. Then, for each sub-strip in order, we push each item of it to the lowest possible position until it touches another item or the bottom of S. Notice that these movements do not affect the

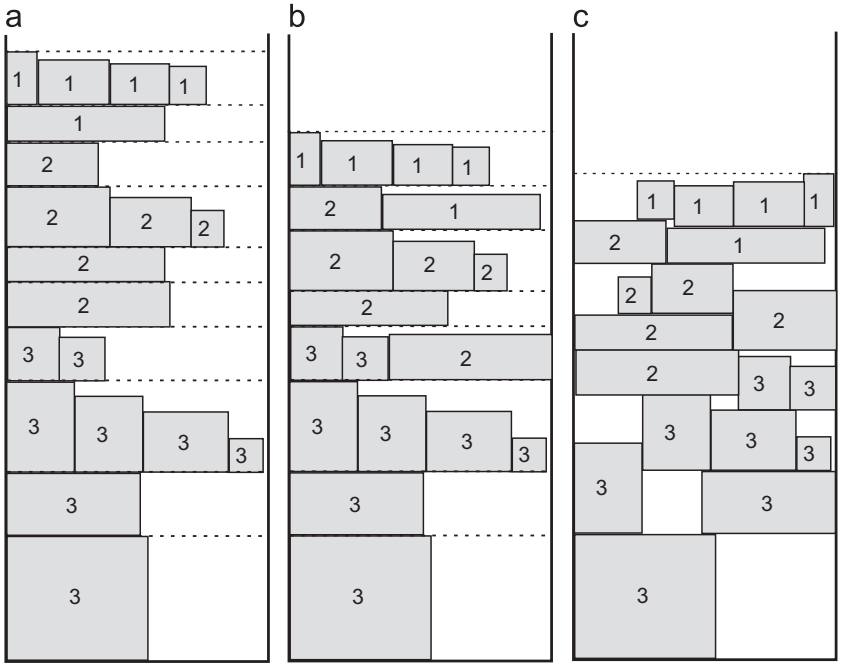

Fig. 3. In this figure we represent the improvements applied on the FFDHC algorithm. In (a) we have the solution computed by the FFDHC algorithm. In (b) we use the first improvement and in (c) the second.

unloading constraints (see Fig. 3 part c). These improvements also do not change the approximation ratio of the FFDHC algorithm.

4. A GRASP based heuristic

In this section we describe our GRASP based heuristic to the SPU problem. First we present a greedy-randomized heuristic to construct initial solutions and then we present a local search procedure that improves the quality of the initial solutions.

4.1. Constructive greedy-randomized heuristic

Our constructive heuristic uses a strategy similar to the one used in [1], but we change the focus of the algorithm to the classes of the items instead of its dimensions. We keep two lists: a list I of items that were not packed yet and a list F of free spaces where items from I can be packed. A free space $f \in F$ is represented by a tuple $(x(f),y(f),w(f),c(f))$ where $(x(f),y(f))$ is the left corner position where the free space starts, $w(f)$ is its width, and $c(f)$ is the minimum class value of some item packed below this free space. The algorithm builds a restricted candidate list (RCL) with items that can be packed in some of the free spaces satisfying the unloading constraint. Then we randomly select one item from the RCL list and pack it at the lowest possible position. This is done until all items are packed, i.e., $I = \emptyset$.

The algorithm works as follows:

*STEP*1 Initialization
- 1.1 Set $I = L$ and $F = \{(0,0,W,\infty)\}$.

*STEP*2: Building the RCL and F'
- 2.1 Let $\text{RCL}^* = \{a_i \in I \,|\, \sum_{a_k \in I | c(a_k) > c(a_i)} w(a_k) \le \rho\}$. This list contains only items that if packed below higher classes items, in principle, they can be removed (they are not blocked by higher classes items).
- 2.2 Notice that each item $a_i \in \text{RCL}^*$ can be packed in some sets of subsequent free spaces. We are interested in the lowest sequence of free spaces where at least one item from RCL* can be packed. So let F' be the set of subsequent free spaces that corresponds to the lowest position where some item from RCL* can be packed.
- 2.3 The final RCL contains the items that can be packed in F':

$$\text{RCL} = \{a_i \in \text{RCL}^* \,|\, a_i \text{ can be packed in a set of subsequent free spaces of } F'\}$$

*STEP*3: Choosing and packing an item
- 3.1 Select an item a_s from RCL with probability $p = w(a_s)/(\sum_{a_k \in RCL} w(a_k))$.
- 3.2 Let $w(F') = \sum_{f \in F'} w(f)$, $y_{max} = max\{y(f) | f \in F'\}$, and $c_{min} = min\{c(f) | f \in F'\}$.
- 3.3 Select the position {left, right} to pack the item a_s into F' at height y_{max} using the procedure showed in [1]. This procedure chooses the x position that leaves as smooth as possible the packing produced, i.e., closest to the sides of the strip S, or closest to the highest border. If $c(a_s) > c_{min}$, go to step 3.1 and select another item, since a_s would block some item of lower class.
- 3.4 Pack a_s into the selected position.

*STEP*4: *Updating the lists*
- 4.1 Update $I = I - a_s$ and F. In F we only need to update the free spaces in F', by updating the minimum items class values, and the dimensions of the free spaces as well.

We also compared other choices of selecting an item in STEP 3. We tried for example to use a probability distribution that considers the classes values combined with the width or height of items. For instance $p = (c(a_i) + kw(a_i))/\sum_{a_k \in RCL} c(a_i) + kw(a_i)$ for some constant k. But the method presented in step 3 outperformed all other choices tested.

4.2. Setting the ρ value

Several experiments were performed in order to discover the best value for ρ (STEP 1). We can see ρ as a factor of aggressiveness of the algorithm, since it allows items of lower classes to be packed before higher classes items. If we use $\rho = \infty$, all items will be in the RCL, while if we use $\rho = 0$, then at each iteration of the algorithm, the items will be selected by decreasing order of class values. Note that the ρ value determines the quality of the RCL list. When ρ is large, the packing of several items in the RCL may be infeasible, while if it is too small we may miss some feasible packings of items in reversed order of class values.

We tried some alternatives to construct the RCL list, despite the one presented in STEP 1 (that uses the ρ value). We performed some experiments to evaluate the following strategies to build the RCL list:

- RCL contains only items from the largest class available (a conservative strategy).
- RCL contains items from the two largest classes available.
- RCL contains items from the largest class available and with probability 50% all the items from the second highest class.

The tests showed that the conservative strategy was the best one. Most of the solutions of the constructive phase were infeasible when using the other two strategies. From that, we could see that items from lower classes should by carefully chosen, since they can easily turn the packing infeasible.

Then we tried the strategy used in STEP 1, that is still aggressive, but avoid the generation of too many infeasible solutions due to the packing of lower class items bellow higher classes items. The different values of ρ used are:

- $\rho = W - w(a_i)$.
- $\rho = (W - w(a_i))/2$.
- $\rho = \lambda(W - w(a_i))$. Where λ is a random value in [0.4,0.6].

These strategies allow items of lower classes to be packed before higher classes items provided that they will probably not generate infeasible solutions, i.e., there is still free space available to remove lower classes items. For instance, when using $\rho = W - w(a_i)$, this imposes that an item is selected only if the sum of the widths of items of higher classes, plus its width is smaller than the width of the strip. In our tests $\rho = (W - w(a_i))/2$ achieved better results consistently, and this is the value used in our algorithm.

4.3. Local search

Given a current solution to the problem, our local search generates three neighbors and the best one is chosen as the new current solution. The process is repeated until no better solution is found. The three neighbor solutions are obtained as follows: given a current solution, remove the last k% items from the packing, where $k \in \{10,20,30\}$. Then, for each one of these three solutions, generate a new one repacking the last k% items using a deterministic constructive algorithm. The deterministic constructive algorithm is an adaptation of the constructive greedy-randomized heuristic, where in step 3 it is always chosen the item with the highest value of p, i.e., the widest item. If a better solution is generated, then the best one replaces the current solution.

We tried to use some other different strategies in the local search. For instance, we tried to impose different ordering of the items like the one in [26,18], before the local search starts. In this case we force the sequence in which items are packed. But the chosen strategy outperforms this one.

5. GRASP heuristic

In this section we present the GRASP heuristic based on the algorithms from Sections 4.1 and 4.3. The GRASP heuristic first finds an initial solution using the constructive algorithm. Then this first solution is improved using the local search procedure. The GRASP heuristic repeats this process until a maximum time is exceeded (60 s in our experiments), and the best solution overall is returned. We denote this algorithm by G.

We implemented another variant of the GRASP heuristic for the case where 90° rotations of items are allowed. In the constructive phase we rotate all items a_i such that $(h(a_i) \geq w(a_i))$ and in the Local Search we rotate the $k\%$ items that are removed and repacked, such that $(h(a_i) \leq w(a_i))$. This strategy is based in the following idea: give more importance to the unloading constraint first and then, in the final part of the packing, give more importance to the total height. At first it seems that we should choose, for each item, the orientation that induces the lowest height. But this strategy produced bad results due to the unloading constraints. Rotating an item to let its width wider, may block several items that could be packed below this item. But in the final part of the packing, the unloading constraint is easier to be satisfied, and that is why we decided to give more importance to the induced height. We note that this strategy achieved better results than some other strategies like: for each solution generated by the constructive algorithm, choose randomly the orientation of the last $k\%$ items and repack them, or use *Reactive* GRASP [29] to choose the orientation of the items. The algorithm for the problem where rotations are allowed is denoted by G^r.

6. Benchmark instances

The algorithms were tested using eight sets of instances. The first set is a classical 2L-CVRP set from the literature (these instances can be downloaded from http://www.or.deis.unibo.it/research.html) without the routing information. The remaining seven sets are instances for the classical Strip Packing Problem that were adapted, by including classes values to items in order to obtain valid instances for the SPU problem.

The first set corresponds to 2L-CVRP instances that were used in [23,19,20,18,26,15]. From these instances we used only the packing data (dimensions of items and Strip) and costumers information (which items belong to which client). The number of customers, items, and their dimensions were created according to five types of instances $c(i)$ (with 36 instances each). For further details about each type of instance $c(i)$, we refer the reader to [23]. This set contains a total of 36×5 instances, containing between 15 and 255 customers and between 15 and 786 items, and we call it *2lcvrp*.

We also adapted seven sets of instances from the classical 2D Strip Packing Problem. For each instance I with n items, we generate five new instances in the following way: let $C_k = \lceil kn/10 \rceil$, for $k = 2,4,\ldots,10$, be the number of classes of each new instance. Then each item from I receives a class uniformly chosen in $[1,C_k]$ (in each respectively new instance). Furthermore, we impose that there exists at least one item for each class $c \in \{1 \ldots C_k\}$. The different number of classes C_k for a same instance helps us study the impact of the number of classes on the size of the generated solution. Notice that the number of classes C_k will be a percentage of the total number of items n, i.e., when $k=2$ for example, C_k corresponds to 20% of n.

Briefly these 7 sets are described bellow:

- *chr*: A set of three instances used by Christofides and Whitlock [12].
- *bke*: A set of 12 instances generated by Burke et al. [9], with 10–500 items.
- *ben*: A set of 10 instances proposed by Bengtsson [7]. Each instance with 25–200 items.
- *htu*: A set of 21 instances proposed by Hopper and Turton [21]. Each instance with 16–197 items.
- *wva*: A set of 420 instances generated by Wang and Valenzuela [31]. This set is partitioned into two subsets: *nice* (with similar shapes and sizes) and *path* (pathological variations on shapes and sizes), with 210 instances each subset, and each instance contains from 25 to 500 items.
- *hop*: A set of 70 instances proposed by Hopper [22]. This set is partitioned into two subsets: T (guillotine) and N (non-guillotine), 35 instances each subset, and each instance with 17 to 199 items.
- *bea*: A set of 25 instances proposed by Beasley [4,6]. This set is partitioned into two subsets: 13 instances (denoted *gcut*) and 12 instances (denoted *ngcut*) with 7–22 items.

Thus we generated 2985 instances to evaluate the GRASP heuristic and the other algorithms. These instances can be obtained at www.loco.ic.unicamp.br/instances/spu2d/.

7. Computational experiments

The algorithms were coded in C and executed on an Intel Core 2 Duo 2.4 GHz processor with 2 GB 667 MHz DDR2 of main memory. The stopping criteria for the GRASP heuristics G and G^r was a time limit of 60 CPU seconds or 1000 iterations (what happens first). We also performed tests with the GRASP heuristics where we limited them to execute five iterations. This was used to evaluate the performance of the heuristics when they need to be executed very fast (as subroutines for the 2L-CVRP problem for example) and to compare its results with the results reported in [18].

Since the GRASP heuristics are non-deterministic we ran it 20 times for each instance and take the average value as its result.

7.1. Lower bounds

To measure the quality of the computed solutions we used the maximum of two lower bounds. One lower bound is based on the total area of items. The other lower bound is based in the unloading constraint. In this case, we use the height of a packing of a subset of the items, that necessarily have to be packed one above the other, due to the unloading constraint.

The second lower bound is computed as follows: we keep a list H, initially empty, of optimal packings of a subset of items. For each item a_i in non-increasing order of class, we do the following: for each packing P in H we decide if the item a_i must be packed above the last packed item in P or not. This decision is made by checking if the width of the last packed item plus the width of the current item, is larger than the strip width. Moreover the class of the last packed item must be larger than the class of the current item. If this is the case, then the current item must be packed above the last packed item, even if there is space to pack the current item bellow the last item. For each packing where a_i must be packed above the last packed item, we update the packing

including a_i. If we could not find any packing where a_i must have to be packed above the last packed item, we create a new packing with item a_i alone, and include this packing in H. Finally, after processing all items, we select the highest packing in H as the lower bound.

For instance, consider the instance N1Burke which has a strip of width 40 and 10 items. One randomly generated instance selecting classes values between [1,10] is presented in Table 1. It is easy to see that the area lower bound of this instance is 40 since $\sum_{i=1}^{10} h(i)w(i)/40 = 40$. However, if we use the second lower bound, we will see that the items a_1, a_4, and a_5 must be packed in order and one above the other due to their widths and the unloading constraint. So we can bound the height by $h(a_1)+h(a_4)+h(a_5) = 46 > 40$ (see Fig. 4 part (a)).

Generally, the second lower bound achieved better lower bound values in instances with items tall and narrow, and small and wide. In some *path* (sub-set of *wva*) instances for example, while the area lower bound achieved a value of 200 the second lower bound achieved a value of 304.

Table 1
Generated random instance from the N1Burke instance.

IItem	$c(a_i)$	$h(a_i)$	$w(a_i)$
a_1	1	6	7
a_2	2	6	7
a_3	3	4	4
a_4	4	16	40
a_5	5	24	24
a_6	6	20	4
a_7	7	20	5
a_8	8	4	5
a_9	9	8	7
a_{10}	10	4	7

7.2. Computational results

In the computational experiments we provide basically two quality measures. The first one is the occupation ratio (occupation column in the tables), which corresponds to the fraction of the used strip that is occupied by items. The second quality measure is the solution ratio (ratio column in the tables), which corresponds to the ratio between the value of the solution found by an algorithm and the value of the best lower bound.

We performed two kinds of tests and measurements:

- Tests with five iterations to measure the occupation. These tests were used to compare our algorithm with the one from [18] (column *Gen* in Table 2).

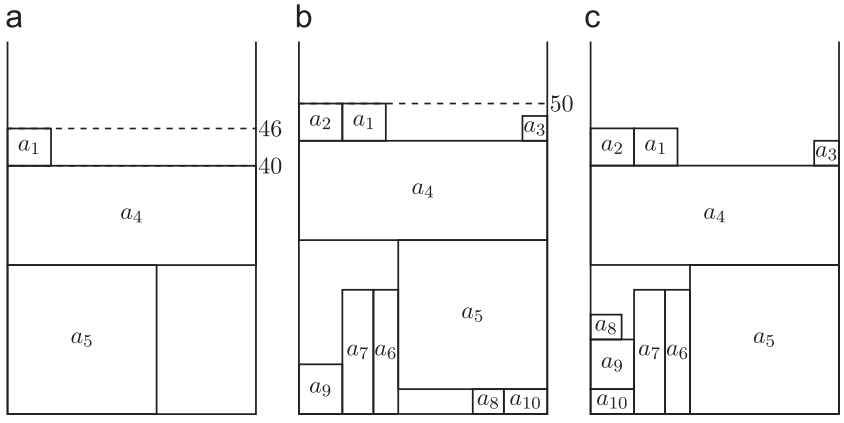

Fig. 4. Generated instance for *N1Burke* with 100% of classes. In (a) we have the lower bound. In (b) we can see the solution found by the algorithm $G(5)$ and in (c) an optimal solution.

Table 2
Results for the *2lcvrp* set.

Set	Type		Occupation			Ratio			
			$G(5)$ (%)	$G^r(5)$ (%)	*Gen* (%)	$G(1000)$	$G^r(1000)$	HBP	$FFDHC^r$
2lcvrp	1	*min*	77.45	77.45	–	1	1	25.9841	1
		ave	77.45	77.45		1	1		
		max	77.45	77.45		1	1		
	2	*min*	77.52	84.01	74.63	1.1495	1.1012	2.3542	1.7590
		ave	80.62	86.26		1.1638	1.1185		
		max	82.33	88.19		1.1832	1.1308		
	3	*min*	80.25	86.32	76.26	1.1390	1.0992	2.2143	1.7017
		ave	82.75	86.70		1.1467	1.1081		
		max	83.95	87.55		1.1588	1.1179		
	4	*min*	82.82	86.29	77.02	1.1355	1.0975	2.0436	1.7022
		ave	83.55	86.91		1.1403	1.1076		
		max	84.24	87.24		1.1455	1.1129		
	5	*min*	85.70	87.98	74.00	1.1265	1.0966	1.9437	1.5506
		ave	85.82	88.13		1.1273	1.0978		
		max	85.98	88.33		1.1299	1.0995		
	Average	*min*	80.74	84.40	75.48	1.1100	1.0789	6.9079	1.5427
		ave	82.03	85.09		1.1152	1.0864		
		max	82.79	85.75		1.1235	1.0922		

- Tests with 1000 iterations and time limit of 60 CPU seconds, where we measured the solution ratio, compared with the lower bounds discussed previously.

In the tables, we denote by $G(5)$ (respectively $G(1000)$) the results for the G heuristic with five iterations (respectively 1000 iterations). Similarly we have $G^r(5)$ and $G^r(1000)$, but for the case where item rotations are allowed. The columns $G(5)$, $G^r(5)$, $G(1000)$, $G^r(1000)$, FFDHC and HBP present the results achieved by each algorithm. For each instance, the grasp heuristics were executed 20 times. We present the minimum, average and maximum values for the G heuristics for these 20 simulations.

In Table 2 we show the results for the set *2lcvrp*. The type column indicates the instance subset. The result for each subset is the average result of all its instances. The first sub-set (1) is a simple generalization of the one dimensional CVRP and does not impose any difficult to the heuristics and they always found the optimal solution. The huge value (25.9841) achieved by the algorithm HBP, occurred because the algorithm just piled the items one above the other.

The $G(5)$ heuristics showed to be a good alternative to be used as a routine within the 2L-CVRP. The heuristics took less than 2 s to find the solutions to the largest *2lcvrp* instances (786 items and 255 customers). They took less then 1 s to solve 80% of the instances approximately. Although they took lower CPU times,

Table 3
Results for the *bea*, *ben* and *bke* sets.

Set	C_k		Occupation		Ratio			
			$G(5)$ (%)	$G^r(5)$ (%)	$G(1000)$	$G^r(1000)$	HBP	FFDHC^r
bea	$k=2$	*min*	74.99	83.04	1.22	1.11	1.60	1.41
		ave	75.73	83.40	1.23	1.13		
		max	76.02	83.68	1.25	1.14		
	$k=4$	*min*	75.36	81.92	1.28	1.15	1.51	1.43
		ave	75.55	82.71	1.30	1.16		
		max	75.97	83.80	1.33	1.19		
	$k=6$	*min*	74.26	80.28	1.25	1.15	1.51	1.46
		ave	74.63	81.35	1.26	1.17		
		max	74.96	82.45	1.28	1.18		
	$k=8$	*min*	71.80	82.20	1.19	1.15	1.46	1.51
		ave	72.08	82.59	1.20	1.17		
		max	73.28	82.87	1.24	1.20		
	$k=10$	*min*	71.92	81.25	1.17	1.16	1.47	1.51
		ave	73.00	82.00	1.18	1.18		
		max	73.78	83.80	1.20	1.22		
	Average	*min*	73.66	81.74	1.22	1.15	1.51	1.46
		ave	74.20	82.41	1.23	1.16		
		max	74.80	83.32	1.26	1.19		
ben	$k=2$	*min*	81.98	84.52	1.06	1.06	1.74	1.37
		ave	86.66	88.41	1.09	1.08		
		max	88.12	90.03	1.17	1.12		
	$k=4$	*min*	82.15	84.46	1.07	1.06	1.74	1.42
		ave	86.95	88.54	1.10	1.07		
		max	88.31	90.50	1.15	1.11		
	$k=6$	*min*	82.44	84.02	1.08	1.07	1.70	1.44
		ave	86.93	88.38	1.10	1.09		
		max	88.75	89.92	1.16	1.14		
	$k=8$	*min*	83.95	84.92	1.08	1.07	1.63	1.51
		ave	87.11	87.99	1.10	1.08		
		max	89.21	89.95	1.15	1.14		
	$k=10$	*min*	83.44	84.54	1.09	1.07	1.67	1.52
		ave	86.29	88.14	1.10	1.09		
		max	88.94	89.09	1.17	1.16		
	Average	*min*	82.79	84.49	1.08	1.07	1.69	1.45
		ave	86.79	88.29	1.10	1.08		
		max	88.67	89.90	1.16	1.13		
bke	$k=2$	*min*	69.43	79.40	1.23	1.12	2.06	1.40
		ave	72.87	81.33	1.26	1.14		
		max	74.22	83.43	1.30	1.16		
	$k=4$	*min*	73.40	81.04	1.22	1.13	2.01	1.40
		ave	75.54	82.28	1.23	1.14		
		max	76.82	83.88	1.26	1.16		
	$k=6$	*min*	69.02	80.02	1.22	1.12	1.99	1.43
		ave	71.86	82.92	1.24	1.13		
		max	73.10	83.15	1.27	1.15		
	$k=8$	*min*	74.70	78.00	1.21	1.17	2.03	1.43
		ave	76.77	80.48	1.23	1.18		
		max	77.98	82.37	1.26	1.20		
	$k=10$	*min*	71.50	79.08	1.22	1.15	2.03	1.44
		ave	74.02%	81.75%	1.24	1.16		
		max	77.91	83.33	1.27	1.17		
	Average	*min*	71.61	79.51	1.22	1.14	2.03	1.42
		ave	74.21	81.75	1.24	1.15		
		max	76.01	83.23	1.27	1.17		

Table 4
Results for the *chr*, *hop* and *htu* sets.

Set	C_k		Occupation		Ratio			
			$G(5)$ (%)	$G^r(5)$ (%)	$G(1000)$	$G^r(1000)$	HBP	FFDHC^r
chr	$k=2$	*min*	81.12	83.02	1.13	1.08	2.01	1.29
		ave	81.24	83.30	1.13	1.08		
		max	81.34	83.48	1.13	1.08		
	$k=4$	*min*	83.80	83.88	1.13	1.10	1.96	1.34
		ave	83.92	83.98	1.13	1.10		
		max	84.08	84.10	1.13	1.10		
	$k=6$	*min*	82.55	84.49	1.14	1.12	1.86	1.39
		ave	82.74	84.78	1.14	1.12		
		max	82.90	84.98	1.14	1.12		
	$k=8$	*min*	81.90	82.00	1.14	1.11	1.58	1.36
		ave	82.06	82.18	1.14	1.11		
		max	82.22	82.29	1.14	1.11		
	$k=10$	*min*	78.88	82.80	1.14	1.15	1.63	1.36
		ave	79.17	83.01	1.14	1.15		
		max	79.38	83.15	1.15	1.16		
	Average	*min*	81.65	83.24	1.14	1.11	1.81	1.35
		ave	81.82	83.42	1.14	1.11		
		max	81.98	83.54	1.14	1.11		
hop	$k=2$	*min*	75.03	80.29	1.18	1.13	2.04	1.41
		ave	77.15	82.42	1.19	1.14		
		max	78.99	83.88	1.22	1.16		
	$k=4$	*min*	74.88	80.96	1.20	1.14	2.06	1.45
		ave	76.03	82.14	1.21	1.15		
		max	77.80	83.12	1.24	1.16		
	$k=6$	*min*	73.22	80.73	1.21	1.14	2.03	1.49
		ave	75.50	82.27	1.23	1.15		
		max	77.07	84.00	1.26	1.17		
	$k=8$	*min*	73.33	79.94	1.21	1.13	2.00	1.51
		ave	75.23	81.86	1.23	1.15		
		max	77.61	83.07	1.26	1.19		
	$k=10$	*min*	71.80	80.13	1.22	1.13	2.03	1.52
		ave	74.20	82.30	1.25	1.16		
		max	75.88	83.99	1.27	1.20		
	Average	*min*	73.98	80.71	1.21	1.14	2.03	1.48
		ave	75.62	82.20	1.22	1.15		
		max	76.16	83.38	1.24	1.17		
htu	$k=2$	*min*	73.99	82.06	1.16	1.09	2.13	1.44
		ave	76.75	84.09	1.18	1.10		
		max	77.81	85.37	1.23	1.14		
	$k=4$	*min*	72.88	81.04	1.19	1.11	2.17	1.47
		ave	75.37	83.58	1.21	1.12		
		max	76.76	84.44	1.24	1.14		
	$k=6$	*min*	73.15	82.00	1.18	1.10	2.13	1.48
		ave	75.78	83.87	1.20	1.11		
		max	77.20	85.12	1.24	1.14		
	$k=8$	*min*	73.06	81.97	1.20	1.10	2.24	1.53
		ave	75.56	83.96	1.22	1.11		
		max	76.99	85.24	1.27	1.13		
	$k=10$	*min*	74.00	81.88	1.20	1.12	2.16	1.56
		ave	75.89	83.23	1.22	1.13		
		max	76.92	84.53	1.25	1.14		
	Average	*min*	73.82	82.11	1.19	1.11	2.16	1.50
		ave	75.87	83.75	1.21	1.11		
		max	76.92	84.74	1.24	1.13		

they achieved a good average occupation on these instances even in the case without rotations, which achieve 82.03% of occupation on average. The results showed in [18] are presented in the column *Gen* of Table 2. To the best of our knowledge there are no other practical results for the SPU problem. Although the main objective in [18] is to minimize the total cost of generated routes, and so the packing problem is biased to pack items in several bins, we think it is interesting to compare their algorithm with ours. Nonetheless these comparisons must be taken carefully since our algorithms are specialized for the SPU problem, and their algorithm is evaluated in the 2L-CVRP scenario. When comparing the average occupation ratio of these algorithms, we can see that the $G(5)$ heuristics achieved better results consistently.

Table 5
Results for the *wva* set.

Set	C_k		Occupation		Ratio			
			$G(5)$ (%)	$G^r(5)$ (%)	$G(1000)$	$G^r(1000)$	HBP	FFDHC^r
wva	$k=2$	*min*	68.99	75.22	1.24	1.17	2.06	1.52
		ave	73.13	78.61	1.27	1.19		
		max	75.29	80.01	1.33	1.23		
	$k=4$	*min*	69.12	75.88	1.24	1.17	2.07	1.55
		ave	73.22	78.71	1.28	1.20		
		max	75.60	75.23	1.32	1.23		
	$k=6$	*min*	68.55	75.12	1.24	1.19	2.06	1.55
		ave	72.92	78.50	1.28	1.21		
		max	74.98	80.22	1.34	1.24		
	$k=8$	*min*	68.30	75.03	1.22	1.18	2.07	1.57
		ave	72.99	78.51	1.28	1.21		
		max	75.18	80.81	1.35	1.25		
	$k=10$	*min*	68.22	75.80	1.26	1.19	2.06	1.57
		ave	72.45	78.30	1.30	1.21		
		max	74.49	80.10	1.34	1.24		
	Average	*min*	69.35	75.93	1.25	1.18	2.06	1.55
		ave	72.94	78.53	1.28	1.20		
		max	74.75	79.15	1.33	1.23		

Table 6
Average time to find the best solution in the *wva* set.

C_k	Num. items	Total time (s)		Time to find the best (s)	
		$G(1000)$	$G^r(1000)$	$G(1000)$	$G^r(1000)$
$k=2$	*25*	1	1	0	0
	50	1	1	0	1
	100	3	4	1	2
	200	13	16	8	10
	500	60	60	36	48
	Average	15.6	16.4	9	12.2
$k=4$	*25*	1	1	0	0
	50	1	1	0	1
	100	3	4	1	2
	200	13	15	7	10
	500	60	60	35	45
	Average	15.6	16.2	8.6	11.6
$k=6$	*25*	1	1	0	0
	50	1	1	0	1
	100	3	4	1	1
	200	13	15	7	8
	500	60	60	33	42
	Average	15.6	16.2	8.2	10.4
$k=8$	*25*	1	1	0	0
	50	1	1	0	1
	100	3	3	1	1
	200	12	14	5	6
	500	60	60	29	39
	Average	15.4	15.8	7	9.4
$k=10$	*25*	1	1	0	0
	50	1	1	0	0
	100	3	3	1	1
	200	12	14	5	6
	500	60	60	28	39
	Average	15.4	15.8	6.8	9.2

Table 7
Results for the 2L-CVRP problem using the heuristics $G(5)$ and *Gen*.

Instance	$G(5)$		*Gen*	
	Cost	Time (s)	Cost	Time (s)
2l-cvrp0101	244	0.02	244	0.02
2l-cvrp0102	264	6.86	–	–
2l-cvrp0103	293	433.59	–	–
2l-cvrp0104	286	63.08	–	–
2l-cvrp0105	272	3.81	–	–
2l-cvrp0201	280	0.01	280	0.01
2l-cvrp0202	304	1479.81	321	3600.00
2l-cvrp0203	302	96.32	315	3598.18
2l-cvrp0204	286	1.54	303	3597.72
2l-cvrp0205	284	0.1	289	20.28
2l-cvrp0301	288	0.03	288	0.03
2l-cvrp0303	389	3600.0	–	–
2l-cvrp0304	377	3600.0	–	–
2l-cvrp0305	334	9.27	–	–
2l-cvrp0401	342	0.03	342	0.03
2l-cvrp0403	379	3597.6	–	–
2l-cvrp0404	403	3600.0	–	–
2l-cvrp0405	358	44.11	–	–
2l-cvrp0501	311	0.05	311	0.06
2l-cvrp0502	445	3600.0	–	–
2l-cvrp0503	376	3600.0	–	–
2l-cvrp0504	372	3600.0	–	–
2l-cvrp0505	357	13.42	–	–
2l-cvrp0601	354	0.04	354	0.05
2l-cvrp0603	472	3600.0	–	–
2l-cvrp0605	374	8.45	–	–
2l-cvrp0701	508	0.03	508	0.04
2l-cvrp0704	674	128.82	–	–
2l-cvrp0705	680	19.5	–	–
2l-cvrp0801	591	0.03	591	0.03
2l-cvrp0802	708	3600.0	–	–
2l-cvrp0804	710	3600.0	–	–
2l-cvrp0805	658	169.96	748	3600.0
2l-cvrp0901	465	0.05	465	0.06
2l-cvrp0903	522	3599.25	–	–
2l-cvrp0905	517	3600.0	–	–
2l-cvrp1001	396	0.27	396	0.26
2l-cvrp1101	417	0.22	417	0.2
2l-cvrp1201	424	0.04	424	0.04
2l-cvrp1203	526	3600.00	–	–
2l-cvrp1205	516	3603.00	–	–
2l-cvrp1301	1853	0.62	1853	0.63
2l-cvrp1305	2691	3600.00	–	–
2l-cvrp1401	657	0.52	657	0.52
2l-cvrp1405	1250	3600.00	–	–
2l-cvrp1501	763	0.56	763	0.56
2l-cvrp1601	506	0.19	506	0.2
2l-cvrp1603	646	3600.0	–	–
2l-cvrp1605	533	774.53	–	–
2l-cvrp1701	598	0.43	598	0.43
2l-cvrp1703	696	3599.72	–	–
2l-cvrp1705	647	3599.46	–	–
2l-cvrp1801	614	2.2	614	2.19
2l-cvrp1901	450	3.15	450	3.48
2l-cvrp2001	212	18.94	212	18.96
2l-cvrp2101	577	6.71	577	56.16
2l-cvrp2201	591	4.19	592	21.59
2l-cvrp2301	626	9.1	626	17.42
2l-cvrp2401	708	30.43	708	47.03
2l-cvrp2501	687	70.03	687	1201.79
2l-cvrp2601	687	190.18	687	188.97
2l-cvrp2701	809	44.64	809	49.29
2l-cvrp2801	552	289.98	–	–
2l-cvrp2901	731	1438.49	–	–
2l-cvrp3001	783	187.88	–	–

The tests with the $G(1000)$ heuristics were stopped by the time limit on 16% of the *2lcvrp* instances approximately. The $G(1000)$ heuristics achieved good solution ratios, considering that we are using lower bounds in these comparisons.

Among the approximation algorithms, the FFDHC^r achieved the best results and all of them took less then 1 CPU second to be computed.

Tables 3–5 contain the results for the remaining instance sets. Remember that for each instance, we created new instances including classes values as explained in Section 6. Since we have a large number of instances, the results presented in these tables corresponds to the averages among all instances of each set, considering a given value k used to derive the number of classes C_k. This way we could study the impact of the number of classes (costumers) and the quality of solutions.

We can see that in most of the instance sets, the average occupation decreases and the ratio increase when the number of classes increases. The largest impact occurs on the FFDHC^r algorithm whose quality of the solutions is proportional to the number of classes. In the *htu* set, while with $k=2$ the FFDHC^r achieved an average ratio of 1.44, with $k=10$ the average ratio achieved was 1.56. Its interesting to note that despite achieving the worst results among all the algorithms, the HBP algorithm is not as heavily affected by the increase in the number of classes, as are the other algorithms.

Among the GRASP heuristics, the worst results were obtained with the *wva* set of instances, while the best ones were obtained with the *ben* instances. We believe that the bad results for the *wva* set occurred mainly because of the weakness of our lower bounds that are used in the comparisons. The heuristic G got, for some instances in the *wva* set, a ratio of 1.4512. In other instances, the heuristic G achieved a ratio of 1.0164, although only 64.47% of occupied area.

In Fig. 4(b and c) we can see an example which shows that the optimal occupation would be 86.45% and that the heuristic G founded a solution with 80.00% of occupation in 5 iterations.

For the HBP algorithm, the best results occurred with the *bea* set, and the worst results occurred with the *htu* set. As for the *2lcvrp* set, the FFDHC^r algorithm achieved consistently the best results among the approximation algorithms.

Table 6 shows the average time used by G and G^r heuristics until they found the best solution for each group of instances of the *wva* set. The tables are organized as follows: the instances are divided by number of classes and items. For each number C_k of classes the table shows the average time for instances with the same number of items. Only the instances with 500 items where stopped by the time limit. Instances with 200 items took about 15.6 s on average. It is possible to see that the time to find the best solution decreases with the increase in the number of classes.

7.2.1. Results for the CVRP problem with two-dimensional loading constraints

We also performed some tests considering the 2L-CVRP problem. We implemented the ILP formulation presented in [19] and used the CPLEX 12.3 solver for solving the problem. For each optimal route found, we used a heuristic to validate the route, i.e, to check if the items of costumers along the route can be packed on the truck satisfying the unloading constraints. If the heuristic could pack the items of the costumers of the route, then we have a feasible route. Otherwise we add a cut in the model removing the current route and re-optimize the model. For sake of simplicity we did not use the capacity constraints while solving the routing problem. We only used the packing constraints in the problem. We also used a time limit of 60 min.

We run the 2L-CVRP ILP model using two different packing heuristics to check the feasibility of routes: one was our best heuristic for the oriented packing problem ($G(5)$), and the other one was the heuristic (*Gen*) proposed in [18]. In Table 7 we show the solutions found when using each heuristic. The heuristic $G(5)$ achieved better results consistently. Using the $G(5)$ heuristic the solver was able to find solutions for 65 instances against 32 when using the heuristic *Gen*. In the instances that were solved by both heuristics, the $G(5)$ found better solutions faster. In all instances of type other than 01, the $G(5)$ achieved solutions with lower cost.

8. Conclusions

In this paper we proposed a new GRASP heuristic and two new approximation algorithms for the Strip Packing Problem with Unloading Constraints (SPU). This problem was previously studied as part of the Two Dimensional Capacitated Vehicle Routing Problem with unload constraints (2L-CVRP). To our knowledge this is the first work to provide a practical study specifically for the SPU problem.

Our GRASP heuristic was based on a well known GRASP heuristic for the Strip Packing problem. We did several adaptations on it to consider the specificities of the SPU problem. We proposed new methods to create the *restricted candidates list* (RCL) considering the particularities of the SPU problem. We tested several possibilities aiming to choose the best strategy to built the RCL. We also adapted the local search procedure to consider the unloading constraints.

We proposed a new approximation algorithm for the problem that is based on a bin packing strategy. Another approximation algorithm was proposed and it is based on the well know FFDH algorithm.

We performed several tests to assess the quality of the solutions computed by the proposed algorithms. A set of tests were done to compare the GRASP heuristics with another algorithm of the literature. The GRASP heuristics obtained better results and can be a good alternative to be used in the 2L-CVRP problem, which was showed in the tests in the 2L-CVRP problem.

In other tests we compared the results of our algorithms with two lower bounds. From the results we could see that the best approximation algorithm was the FFDHC^r. But overall, the GRASP heuristics outperformed all algorithms obtaining very good results.

References

[1] Alvarez-Valdez R, Pareño F, Tamarit JM. Reactive GRASP for the strip-packing problem. Computers & Operations Research 2008;35:1065–83.
[2] Augustine J, Banerjee S, Irani S. Strip packing with precedence constraints and strip packing with release times. Theoretical Computer Science 2009;410(38-40):3792–803.
[3] Azar Y, Epstein L. On two dimensional packing. Journal of Algorithms 1997;25(2):290–310.
[4] Beasley JE. Algorithms for unconstrained two-dimensional guillotine cutting. Journal of the Operational Research Society 1985;36:297–306.
[5] Belov G, Scheithauer G, Mukhacheva EA. One-dimensional heuristics adapted for two-dimensional rectangular strip packing. Journal of the Operational Research Society 2008;59:823–32.
[6] Beasley JE. An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research 1985;33:49–64.
[7] Bengtsson BE. Packing rectangular pieces—a heuristic approach. The Computer Journal 1982;25:353–7.
[8] Bortfeldt A. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. European Journal of Operational Research 2006;172:814–37.
[9] Burke EK, Kendall G, Whitwell G. A new placement heuristic for the orthogonal stock-cutting problem. Operations Research 2004;52:655–71.
[11] Burke EK, Hyde M, Kendall G. A squeaky wheel optimisation methodology for two dimensional strip packing. Computers & Operations Research 2010;38:1035–44.

[12] Christofides N, Whitlock C. An algorithm for two-dimensional cutting problems. Operations Research 1977;25:31–44.
[14] Coffman Jr. EG, Garey MR, Johnson DS, Tarjan RE. Performance bounds for level-oriented two-dimensional packing algorithms. SIAM Journal on Computing 1980;9(4):808–26.
[15] Doerner KF, Fuellerer G, Hartl RF, Iori M. Ant colony optimization for the two-dimensional loading vehicle routing problem. Computers and Operations Research 2009;36:655–73.
[16] Fekete SP, Kamphans T, Schweer N. Online square packing. Algorithms and Data Structures LNCS 2009;5664:302–14.
[18] Gendreau M, Iori M, Laporte G, Martello S. A Tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. Network 2008;51:4–18.
[19] Iori M, Salazar Gonzalez JJ, Vigo D. An exact approach for vehicle routing problems with two-dimensional loading constraints. Technical report OR/03/04, DEIS, University of Bologna; 2003.
[20] Iori M, Salazar Gonzalez JJ, Vigo D. An exact approach for the vehicle routing problem with two-dimensional loading constraints. Transportation Science 2007;41:253–64.
[21] Hopper E, Turton BCH. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. Artificial Intelligence Review 2001;16:257–300.
[22] Hopper E, Turton BCH. Problem generators for rectangular packing problems. Studia Informatica Universalis 2002;2:123–36.
[23] Iori M, Martello S, Monaci M. Metaheuristic algorithms for the strip packing problem. In: Pardalos P, Korotkich V. editors. Optimization and industry: new frontiers; 2003. pp. 159–79.
[25] Iori M, Martello S. Routing problems with loading constraints. TOP 2010;18:4–27.
[26] Kiranoudis CT, Tarantilis CD, Zachariadis EE. A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. European Journal of Operational Research 2009;195:729–43.
[27] Lodi A, Martello S, Vigo D. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. INFORMS Journal on Computing 1999;11:345–57.
[28] Miyazawa FK, Wakabayashi Y. Parametric on-line algorithms for packing rectangles and boxes. European Journal of Operational Research 2003;150:281–92.
[29] Prais M, Ribeiro CC. Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment. INFORMS Journal on Computing 2000;12:164–76.
[30] Toth P, Vigo D. The vehicle routing problem. SIAM monographs on discrete mathematics and applications, Philadelphia; 2002.
[31] Valenzuela CL, Wang PY. Data set generation for rectangular placement problems. European Journal of Operational Research 2001;134:378–91.