WILEY

INTERNATIONAL
TRANSACTIONS
IN OPERATIONAL
RESEARCH

Intl. Trans. in Op. Res. 23 (2016) 355–368
DOI: 10.1111/itor.12107

Recursive circle packing problems

João Pedro Pedroso[a,b], Sílvia Cunha[a,c] and João Nuno Tavares[a,c]

[a]*Faculdade de Ciências, Universidade do Porto, Rua Campo Alegre, 1021/1055, 4169-007 Porto, Portugal*
[b]*INESC Porto, Rua Dr. Roberto Frias, 378, 4200-465 Porto, Portugal*
[c]*Centro de Matemática da Universidade do Porto, Rua do Campo Alegre, 687, 4169-007 Porto, Portugal*
E-mail: jpp@fc.up.pt [Pedroso]; silvia.cunha@fc.up.pt [Cunha]; jntavar@fc.up.pt [Tavares]

Received 29 July 2013; accepted 13 May 2014

Abstract

This paper presents a class of packing problems where circles may be placed either inside or outside other circles, the whole set being packed in a rectangle. This corresponds to a practical problem of packing tubes in a container. Before being inserted in the container, tubes may be put inside other tubes in a recursive fashion. A variant of the greedy randomized adaptive search procedure is proposed for tackling this problem, and its performance is assessed in a set of benchmark instances.

Keywords: packing problems; knapsack problems; heuristics; practice of OR; combinatorial optimization; integer programming; loading problems; local search

1. Introduction

Recursive circle packing problems (RCPP) originated in the tube industry where shipping costs represent an important fraction of the total cost of product delivery. Tubes are produced in a continuous extraction machine and cut to the length of the container inside which they will be shipped. Before being placed in the container they may be inserted inside other, thicker tubes, so that usage of container space is maximized—a process named “telescoping.” As all the tubes occupy the full container length, maximizing container load is equivalent to maximizing the area filled with circles (or, more precisely, rings/annuli) in a section of the container.

This problem is more general than circle packing, which is known to be NP-complete (see, e.g., Lenstra and Rinnooy Kan, 1979). In this paper, we propose a heuristic method for tackling it, which has proved to be able to produce very good solutions for practical purposes.

The main contributions of this paper are the following:

- Presentation and formalization of new, recursive circle packing problems (RCPP).
- Development of a variant of the greedy randomized adaptive search procedure (GRASP) for tackling it.

Published by John Wiley & Sons Ltd, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main St, Malden, MA02148, USA.

- Selection of a set of interesting and challenging benchmark instances.
- Experimental analysis of the solution method proposed.

This paper is organized as follows. In Section 2, we present a summary of the literature relevant to RCPP. We then formalize a description of the problem in mathematical terms in Section 3, and present a method for tackling it in Section 4. An experimental analysis of its performance is provided in Section 5, and conclusions and directions for future research are drawn in Section 6.

2. Background

No previous references to RCPP, as described in this paper, could be found. However, the literature presents several problems with similarities.

A nontechnical, general overview of circle packing is presented in Stephenson (2003), in an easy to read and captivating way. For an interesting and useful bibliographic review article see Hifi and M'Hallah (2009) that surveys the most relevant literature on efficient models and methods for packing circular objects/items into regions in the Euclidean plane; objects/items and regions considered are either two- or three-dimensional. A survey of industrial applications of circle packing and of methods for their solution, both exact and heuristic, is presented in Castillo et al. (2008).

The so-called "strip packing problem," as well as the knapsack problem, is solved in Kubach et al. (2009). The first problem relates to a placement of unequal circles within a rectangular strip of fixed width so that its variable length is minimized, whereas the knapsack problem requires a packing of the circles in a fixed size rectangle. They solve both problems using a greedy algorithm that enhances algorithms used in Huang et al. (2004), and is tested in instances from Stoyan and Yaskov (2004). A different approach has been proposed in Stoyan and Yaskov (2014), where unequal circles are packed into a rectangular strip by assuming radii of circles to be variables; the solution process involves decreasing the problem's dimension by fixing radii of some circles and rearranging certain pairs of circles.

The problem of finding the smallest object within which a set of items can be packed has been dealt with in Birgin and Sobral (2008), where circular, triangular, squared, rectangular, and strip objects are considered both in two and in three dimensions. This work presents twice-differentiable models for these problems, allowing the solution of instances with a large number of items.

A model for the problem of packing both circles and nonoriented nonconvex polygons with rotations into a multiply connected region is presented in Stoyan et al. (2012), where an approximation to a global minimum is obtained by means of a specialized, nonexhaustive search of local minima.

Cutting is a problem often associated with packing, nonlinear programming models for cutting circles from rectangular plates, and techniques to solve them are described in Kallrath (2009).

Nonlinear programming models for packing identical circular items in elliptical objects are presented in Birgin et al. (2013), where multistart strategies are used in connection to nonlinear programming solvers for searching a global solution. Packing of identical circles within triangles, rectangles, and strips is dealt with in Birgin and Gentil (2010), with the aim of minimizing the containers' dimensions. Models and a solution method for packing identical circles into a circle with circular prohibited areas have been presented in Stoyan and Yaskov (2012).

Huang et al. (2004), use a greedy algorithm to pack unequal circles into a rectangle. To place the next circle, they use a performance measure called hole degree, which indicates how close this circle is accommodated between the other already placed circles and the rectangle's sides. This algorithm is easily adapted if, instead of in a rectangle, circles are packed inside a larger circle, as proposed by Huang et al. (2006). These papers also describe the basic placement heuristics considered in this work, except for telescoping.

George et al. (1995) study the problem of fitting circles of different sizes into a rectangle, which is formulated as a nonlinear mixed integer programming problem; they also develop a number of heuristic procedures for solving it. The best performing heuristic methods were a quasi-random technique and genetic algorithm using the concept of stable solution structure.

A simulated annealing method for solving unconstrained and constrained circular cutting problems has been presented in Hifi et al. (2004); the approach used is based on an energy function that, when its value is minimized, leads to solutions composed by a set of pieces concentrated at the bottom-left corner of the initial rectangle.

Typologies of circle and other packing problems can be found, for example, in Hifi et al. (2004) and Wäscher et al. (2007). Despite the similarities of some of the classes with the problem we are dealing with, its recursive nature seems to be appearing here for the first time.

3. Problem statement

3.1. The base model

In the base model, A tubes are available for packing in a container of width W and height H, in such a way that the value of the packing is maximum. Let $\mathcal{A} = \{1, \dots, A\}$ be the index set of the tubes; each tube $i \in \mathcal{A}$ is characterized by an external radius r_i^{ext} and an internal radius r_i^{int}, and may be placed in the container or not. For describing a solution we will use

- binary variables w_i for all $i \in \mathcal{A}$, where $w_i = 1$ if tube i is placed directly inside the container, $w_i = 0$ otherwise;
- binary variables u_{ki} for $k, i \in \mathcal{A}$ such that $r_k^{\mathsf{int}} \geq r_i^{\mathsf{ext}}$, where $u_{ki} = 1$ if tube i is placed directly inside tube k, $u_{ki} = 0$ otherwise (only required if $r_k^{\mathsf{int}} \geq r_i^{\mathsf{ext}}$; other pairs (k, i) are not excluded for facilitating notation);
- position variables (x_i, y_i) of the center of each tube i, for all $i \in \mathcal{A}$ (only relevant if i is placed).

The constraints are the following. Each loaded tube is placed within the bounds of a container, which we assume to be a rectangle with vertices $(0, 0)$, $(W, 0)$, $(0, H)$, and (W, H):

$$x_i - r_i^{\mathsf{ext}} \geq 0, \quad \forall i \in \mathcal{A}, \tag{1}$$

$$y_i - r_i^{\mathsf{ext}} \geq 0, \quad \forall i \in \mathcal{A}, \tag{2}$$

$$x_i + r_i^{\mathsf{ext}} \leq W, \quad \forall i \in \mathcal{A}, \tag{3}$$

$$y_i + r_i^{\text{ext}} \leq H, \quad \forall i \in \mathcal{A}. \tag{4}$$

Loaded tubes may be placed either directly in the container or inside other tubes, in the latter case only if the other tube is itself inserted

$$w_i + \sum_k u_{ki} \leq 1, \quad \forall i \in \mathcal{A}, \tag{5}$$

$$\sum_i u_{ki} \leq w_k + \sum_i u_{ik}, \quad \forall k \in \mathcal{A}. \tag{6}$$

For each pair of tubes (i, j) directly placed in the container, the distance between them must be larger than the sum of their external radii

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq (r_i^{\text{ext}} + r_j^{\text{ext}})(w_i + w_j - 1), \quad \forall i, j \in \mathcal{A}. \tag{7}$$

Note that this constraint is nonredundant if and only if $w_i = w_j = 1$. The same constraint must be observed for each pair of tubes (i, j) directly placed inside the same tube k.

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq (r_i^{\text{ext}} + r_j^{\text{ext}})(u_{ki} + u_{kj} - 1), \quad \forall i, j, k \in \mathcal{A}. \tag{8}$$

If tube i is placed directly in tube k, their centers must be close enough for i to remain completely inside k.

$$\sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \leq r_k^{\text{int}} - r_i^{\text{ext}} + M(1 - u_{ki}), \quad \forall i, k \in \mathcal{A}. \tag{9}$$

Recall that only pairs (k, i) such that $r_k^{\text{int}} \geq r_i^{\text{ext}}$ (i.e., k is large enough to contain i) are considered. The constant M is large enough to make each of this constraint redundant if u_{ki} is zero, but for efficiency should be no larger. The farthest positions for k and i are touching opposite corners in the rectangle; hence we can set $M = \sqrt{(H - r_i^{\text{ext}} - r_k^{\text{ext}})^2 + (W - r_i^{\text{ext}} - r_k^{\text{ext}})^2}$.

Thus, constraints (7)–(9) assure feasibility concerning the relative position of the tubes. Note that these constraints may be squared, in order to preserve differentiability. This would be important for tackling the problem with general purpose nonlinear solvers, in line with what has been done in Birgin et al. (2005).

The objective of this problem is to maximize the value of the packing, that is, the sum of a user-defined value v_i for loaded tubes:

$$\text{maximize } V = \sum_{i \in \mathcal{A}} v_i \left(w_i + \sum_{k \in \mathcal{A}} u_{ki} \right). \tag{10}$$

3.2. *Model for minimization of the number of containers*

The previous model considers that every tube may be packed or not; let us now turn to the case where there are N tubes that must be packed, and let $\mathcal{N} = \{1, \ldots, N\}$ be their index set. It is assumed that C identical containers with width W and height H are available, and their index set is denoted by $\mathcal{C} = \{1, \ldots, C\}$.

We now need the following variables:

- Instead of w_i, binary variables w'_{ci} for all $c \in \mathcal{C}, i \in \mathcal{N}$, where $w'_{ci} = 1$ if tube i is placed directly inside container c, $w'_{ci} = 0$ otherwise.
- Binary variables $z_c = 1$ if container c is used, $z_c = 0$ otherwise, for all $c \in \mathcal{C}$.

Constraints (1)–(4), (8), and (9) must be now verified for all $i \in \mathcal{N}$. Constraints (5) are replaced by

$$\sum_{c\in\mathcal{C}} w'_{ci} + \sum_{k\in\mathcal{N}} u_{ki} = 1, \quad \forall i \in \mathcal{N}, \tag{11}$$

imposing that every tube must be put either in a container or in another tube, and replacing (7) by

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq (r_i^{\text{ext}} + r_j^{\text{ext}})(w'_{ci} + w'_{cj} - 1) \quad \forall i, j \in \mathcal{N}, \forall c \in \mathcal{C}. \tag{12}$$

Additional constraints assert that all containers with tubes inside are counted.

$$z_c \geq w'_{ci} \quad \forall c \in \mathcal{C}, \forall i \in \mathcal{N}. \tag{13}$$

For reducing symmetry, one may additionally impose low indices for the open containers:

$$z_c \leq z_{c-1} \quad \text{for } c = 2, \ldots, C. \tag{14}$$

The objective is now to minimize the number of containers, that is,

$$\text{minimize } C = \sum_{c\in\mathcal{C}} z_c. \tag{15}$$

This model can be extended to the case of nonidentical containers by considering modified versions of (3) and (4); for example, (4) becomes $y_i + r_i^{\text{ext}} \leq \sum_c H_c w'_{ci}$, where H_c is the height of container c. In this case, constraints (14) do not apply, and the objective is to minimize $\sum_c f_c z_c$, where f_c is the cost of using container c.

3.3. *Model for maximization of load value*

A variant of the problem that is more relevant in many situations is that of maximizing container load. It can be stated as follows: determine the minimum number of containers for dispatching N required tubes; then, given A additional, nonrequired tubes, insert a selection of them so that the value sent in the required containers' space is maximum.

The procedure can be summarized in following two steps:

(1) Determine the minimum number of containers C^* necessary for delivering required tubes, using the model of Section 3.2.
(2) Maximize the value for nonrequired tubes that are loaded in these containers.

We will focus on the solution of the second step, which is carried out after having solved the model of Section 3.2. The index sets are now $\mathcal{C}^* = \{1, \ldots, C^*\}$ for required containers, $\mathcal{N} = \{1, \ldots, N\}$ for required tubes, and $\mathcal{A} = \{N+1, \ldots, N+A\}$ for nonrequired tubes; variables w'_{ci} and u_{ki}, for $c \in \mathcal{C}^*$ and $k, i \in \mathcal{N} \cup \mathcal{A}$, have the same meaning as before.

The objective for this model is as follows:

$$\text{maximize } V = \sum_{i \in \mathcal{A}} v_a \left(\sum_{c \in \mathcal{C}^*} w'_{ci} + \sum_{k \in \mathcal{N} \cup \mathcal{A}} u_{ki} \right) \tag{16}$$

subject to constraints (1)–(4), (8), (9), (12), and (13), which must be verified for all $i, j \in \mathcal{N} \cup \mathcal{A}$; constraint (11) is replaced by

$$\sum_{c \in \mathcal{C}} w'_{ci} + \sum_{k \in \mathcal{N} \cup \mathcal{A}} u_{ki} = 1, \quad \forall i \in \mathcal{N}, \tag{17}$$

$$\sum_{c \in \mathcal{C}} w'_{ci} + \sum_{k \in \mathcal{N} \cup \mathcal{A}} u_{ki} \leq 1, \quad \forall i \in \mathcal{A}. \tag{18}$$

Because of its practical significance, this is the model tackled in this paper.

3.4. *Limitations and practical issues*

The main limitation of the previous models concerns the quadratic number of variables u_{ki} that model whether tube i is placed within tube k or not. This is likely to interdict the use of general purpose nonlinear integer optimization solvers for tackling the problem.

Several practical issues have been omitted in the problem description, for the sake of clarity. Besides space occupation, containers also have a weight limit that cannot be exceeded. Some tubes have limited robustness, and load placed above them is limited. Good practical solutions have a low center of gravity, that is, heavier tubes are placed in the bottom of the container. In many situations, tubes have sockets, making them thicker in one of the extremities; in this case, a section of the container is no longer enough to represent a solution.

Optimal solutions are not necessarily practical. The position of certain tubes may be awkward in what concerns container loading, for example if tubes in lower layers are not in a stable, fixed position at the time of loading upper layers. In practice, the decision maker wants to have a set of solutions and intuitively choose one that is easy to implement and satisfactory in terms of space usage.

4. Solution method

4.1. *Placement heuristics*

As an approximative way of tackling this problem we propose that, when packing a tube (i.e., a ring) in the rectangular container, this tube is telescoped with other thinner tubes. Telescoping is done in a recursive fashion, that is, any tube placed inside may itself be filled with available thinner tubes; this is recursively attempted until no other candidates can be inserted. After telescoping is completed, the outer tube (hence, possibly with other tubes inside) is inserted in the container. This corresponds to the more usual problem of packing unequal circles into a rectangular container. The telescoping phase is presented in Algorithm 1, and packing in the rectangular container in Algorithm 2.

Algorithm 1. Recursive tube packing in a tube (telescoping).

```
Input: Outer tube $t$, set of available tubes $\mathcal{L}$; each tube is characterized by its
inner and outer diameter.
Output: Set of tubes inserted in $t$; for each of them, the (recursive) set of tubes
placed inside, and the coordinates of theirs centers.
TELESCOPE($t, \mathcal{L}$)
    let $\mathcal{I} := \{t\}$                          ←set of telescoped tubes
    foreach $s \in \mathcal{L}$        ←tubes are assessed by decreasing outer diameter
        if $s$ can be placed inside $t$:
            let $\mathcal{L} := \mathcal{L} \backslash \{s\}$
            let $\mathcal{I} := \mathcal{I} \cup$ TELESCOPE($s$, $\mathcal{L}$)          ←recursive call
    return $\mathcal{I}$
```

After the telescoping phase is completed on an outer tube, the output is given as input to the packing procedure. This means that the circles that will be packed in each rectangle may have other, smaller circles packed inside (see Fig. 4).

Algorithm 2. Packing circles in a rectangular container.

```
Input: Set of tubes to be packed $\mathcal{L}$; dimensions of the rectangular container.
Output: Set of tubes packed; coordinates of their centers.
RECTANGLEPACKING($\mathcal{L}$)
    let $\mathcal{C}$←:= {}                          ←start with empty container
    foreach $t$ ∈←$\mathcal{L}$        ←tubes are assessed by decreasing outer diameter
        if $t$ can be placed inside the container:
            let $\mathcal{L}$←:= $\mathcal{L} \backslash \{t\}$
            let $\mathcal{C}$←:= $\mathcal{C}$←∪←TELESCOPE($t$, $\mathcal{L}$)
            set $t$'s position to the lowest, leftmost position available
    return $\mathcal{C}$
```

In the two previous algorithms, the basic tool for choosing a place for a circle is a point where it is tangent to two other objects (the exception is the first circle being packed inside a circle; this is placed in the bottom, as shown in Fig. 1). Possible points for placing a circle tangent to two objects are presented in Figs. 1 and 2.

A heuristics for positioning a circle consists of choosing the lowest ordinate among the possible positions for each circle, and for tie-breaking selecting the leftmost among positions with the same (lowest) ordinate. These tools specify a greedy procedure for obtaining solutions for the container minimization version of the RCPP: sequentially they create new containers for packing tubes, inserting in each of them telescoped tubes, until the list of tubes remaining unpacked is empty.

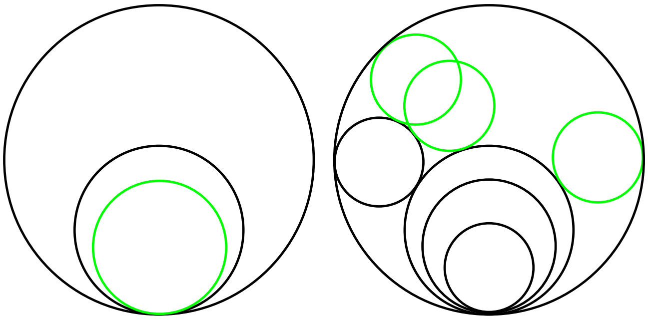

Fig. 1. Possibilities of circle packing inside another circle (telescoping): positioning possibilities given previously placed, fixed circles (in black).

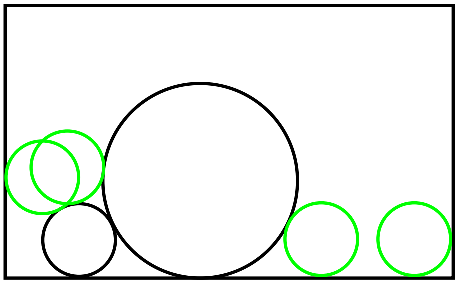

Fig. 2. Circle packing inside a rectangle: positioning possibilities given previously placed, fixed circles (in black).

As for the load maximization version, the procedure is adapted in the following way: after packing a container with required tubes, the container and the tubes within are filled up with nonrequired tubes, again in a greedy fashion. This is done starting from the most valuable nonrequired tube, and attempted until no additional tube can be inserted without overlap, or until the list of nonrequired tubes is empty. New containers are opened only as long as there are unpacked required tubes.

4.2. Local search

Local search in combinatorial optimization allows improving a solution by searching among its neighbors until reaching a local optimum, that is, a solution with no neighbors better than itself. Neighbors are solutions that share most of the structure with the incumbent solution.

If the neighborhood is not carefully designed, changes to the solution provided by the previous construction methods—as occasioned by a local search procedure—may be rather difficult to compute, as they may induce a chain of circle overlaps. In order to keep the algorithms simple and avoid dealing with infeasible solutions, we have used the following local search procedure: for each tube, attempt changing its placement into a lower position not originating overlap.[1] As local search may require a substantial amount of CPU time, it is only done on solutions that improve the best found so far.

4.3. Semigreedy construction and improvement

Greedy construction has the advantage of producing solutions very quickly, and in a very orderly manner. These characteristics are very highly valued in practice, but the quality of greedy solutions may be rather limited. In order to overcome this problem, one straightforward strategy consists of allowing more than one place to be considered as candidate positions for each tube—borrowing ideas from semigreedy construction in GRASP metaheuristics (for an introduction to GRASP see, for example, Feo and Resende, 1995). The steps for doing this, one container at a time, are sketched in Algorithm 3.

Algorithm 3. Semigreedy construction and improvement.

```
Input: Number of iterations $T$; remaining tubes ($\mathcal{R}$←required, $\mathcal{A}$←non-required).
Output: Set of tubes used in this packing; coordinates of their centers.
1   let $c^*$ := greedy solution from tubes in $\mathcal{R}$
2   while $\mathcal{R}$←not empty:
3       complete $c^*$ greedily with items removed from $\mathcal{R}$
4       let $\mathcal{L}$←:= list of tubes in $^*c$
5       add next tube from $\mathcal{R}$←to $\mathcal{L}$
6       for $1$ to $T$
7           use Algorithm 2 for packing $\mathcal{L}$, excepting:
8               for each tube, determine a list $\mathcal{C}$←of candidate positions
9               choose its position randomly from $\mathcal{C}$
10          construct solution $c'$ with these steps
11          if all tubes in $\mathcal{L}$←were packed into $'c$ :
12              execute a local search descent on $c'$
13              let $c^*$ := $c'$
14              update $\mathcal{R}$
15              continue from step (2)
16      break
17  (repeat steps (2) to (16) with list $\mathcal{A}$←instead of $\mathcal{R}$)
18  return set of tubes inside $c^*$ and their positions
```

In step (8) of Algorithm 3 a so-called “restricted candidate list” $\mathcal{C}$ of the best placement positions is prepared for each tube; the actual position in the current construction is chosen randomly from this list. For the RCPP, the number of potential positions for a tube is rather difficult to estimate. Instead of limiting the size of the restricted candidate list, it is preferable to keep all the feasible candidates, sort the list according to the placement heuristics, and then randomly choose an item with decreasing probability with respect to its rank.

---

[1] This is particularly important in practical situations, where load weight must be considered and solutions with a low center of gravity are preferred.

Using this randomized greedy procedure, steps (7) to (10) attempt to insert the current set of tubes $\mathcal{L}$ in one container. If this is successful, a new item is added to the list of tubes to be inserted, in step (5), and the process is repeated. When in T iterations the list of tubes could not be put in a container, the current found solution is accepted. Semigreedy filling of the container with an increasing number of nonrequired tubes are then attempted, using the same steps. The current usage of Algorithm 2 in step (7) may make use of semigreedy placement of tubes inside tubes, as is being done for placing tubes in the container. This has been attempted, but for the instances considered success was limited; therefore, recursive packing in tubes is done in a purely greedy fashion.[2]

As the number T of attempted solutions increases, the quality of the best-found solution tends to improve; T must be set according to the time available and the speed of the computer being used.

Note that even though this algorithm was designed with the load maximization version of the problem in view, it can be used for the container minimization version in a straightforward way by setting $\mathcal{A}$ to an empty list.

4.4. *The complete solution method*

The complete solution method can now be summarized in Algorithm 4. As mentioned, it can be classified as a variant of GRASP; the main idea is to construct a complete solution sequentially, one container at a time. In each iteration of the main cycle, items remaining in the lists of required and nonrequired tubes are used for calling Algorithm 3 and filling an additional container. The lists of remaining required and nonrequired tubes are then updated. This cycle is repeated until there are no more required tubes. The solution returned consists of the set or containers obtained, each of which characterized by the set of tubes inside it and their positions.

Algorithm 4. GRASP for RCPP.

```
Input: Number of iterations $T$; lists of required ($\mathcal{R}$) and non-required ($\mathcal{A}$) tubes.
Output: Set of containers; set of tubes in each container and its coordinates.
let $\mathcal{C}$←:= set of containers (initially empty)
while $\mathcal{R}$←not empty:
    let $c$ := semi-greedy construction with $\mathcal{R}$,$\mathcal{A}$, $T$
    remove tubes inserted in $c$ from $\mathcal{R}$,$\mathcal{A}$
    let $\mathcal{C}$←:= $\mathcal{C}$←∪←$\{c\}$
return $\mathcal{C}$
```

Practical instances of this problem have many identical items. When the number of containers is large, some of them are typically an exact copy of their predecessor. For speeding up the algorithm in this case, before attempting semigreedy construction of a container's contents, it is checked if there are enough remaining items to fill it exactly like its predecessor; if so, the predecessor's configuration is accepted for the current container.

---

[2] For instances allowing a large number of tubes inside other tubes, semigreedy recursive packing in tubes is likely to be worthy.

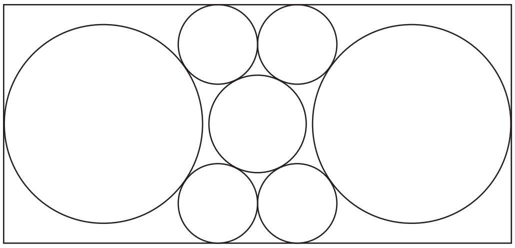

Fig. 3. A solution that cannot be found by the proposed methods.

4.5. Optimality

One question that may be asked about the proposed solution method concerns its convergence to optimality in case the number of construction iterations is unlimited. The answer is negative; indeed, in examples as the one shown in Fig. 3, the sequential placement of circles is not on positions where they touch two other objects. Small and middle tubes in the central-top and bottom positions hinder the movement of large tubes. Such a solution cannot be found by the methods proposed in this paper.

5. Computational results

In order to assess the quality of the methods proposed, we performed a computational experiment using instances similar to those found in an industrial setting. The methods were extensively tested on a set of nine challenging instances, divided into three groups, each characterized by the number of different tubes. For each group, the first (smallest) instance has been found in situations where a greedy solution was not satisfactory: the set of required tubes can be loaded in one container,

Table 1
Number of containers required (C), value or nonrequired tubes inserted (V), and CPU time (seconds) used for each of the test instances, as obtained by greedy construction and by the GRASP metaheuristics (for the latter, minimum, median, and maximum values on 25 independent observations).

				GRASP						
	Greedy			Worst		Median		Best		Average
Instance	C	V	CPU	C	V	C	V	C	V	CPU
`s03i1`	2	407	0.45	1	20	1	23	1	29	14
`s03i2`	11	570	1.5	10	204	10	252	10	300	215
`s03i3`	103	3031	23.	98	1807	96	930	95	1034	323
`s05i1`	2	532	0.40	1	38	1	41	1	43	11
`s05i2`	11	838	1.7	10	435	10	440	10	445	338
`s05i3`	101	4188	15	99	3858	99	3955	98	3577	465
`s16i1`	2	5069	1.3	1	922	1	942	1	966	25
`s16i2`	10	9616	4.4	10	9450	10	9598	10	9664	1084
`s16i3`	98	88,062	78	97	83,525	96	79,641	96	80,613	2773

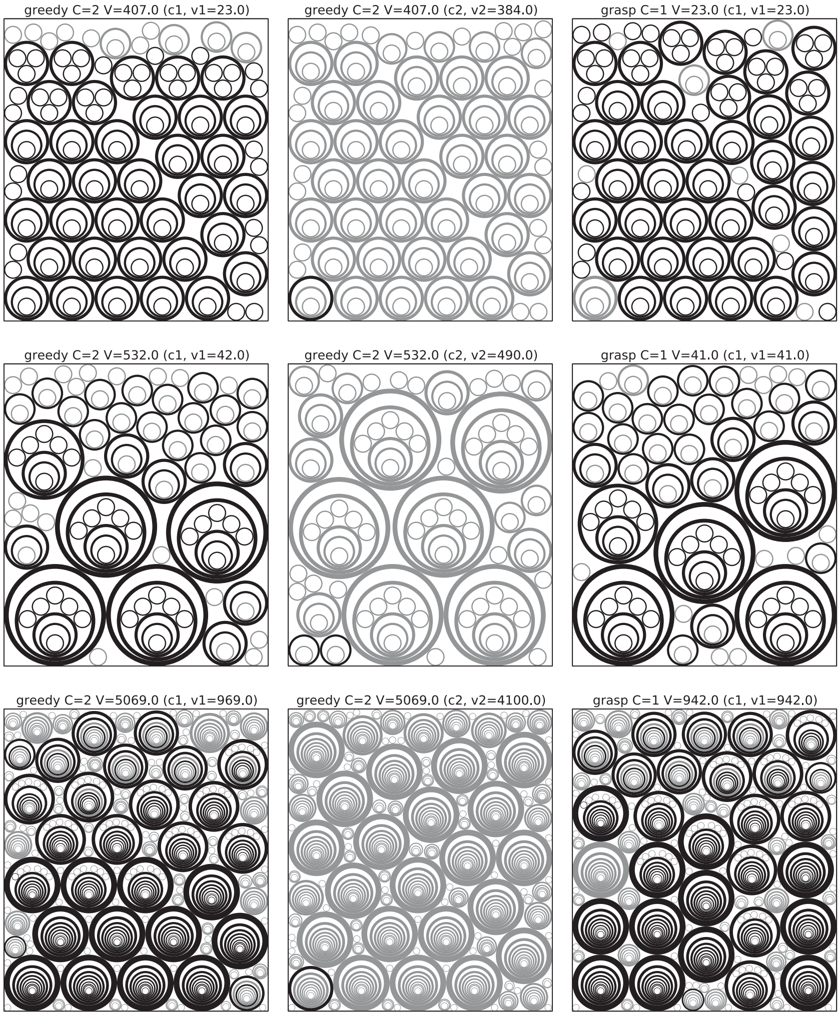

Fig. 4. Plots of solutions obtained with greedy construction (left and center) and with GRASP (median of solutions obtained in 25 independent observations, right) for the small instances: `s03i1` (top), `s05i1` (center), and `s16i1` (bottom). Required tubes are plotted in black, and nonrequired ones in gray.

but the greedy solution uses two. The remaining instances of each group are obtained multiplying the number of required tubes by 10 and 100—hence leading to an expected approximate number of containers of 10 and 100, respectively.[3]

The first group of instances, `s03i1`, `s03i2`, `s03i3`, corresponds to tubes of three different sizes; group `s05i1`, `s05i2`, `s05i3` has five different tube sizes, and the third group, `s16i1`, `s16i2`, `s16i3`, consists of instances with 16 different tube sizes.

Results obtained in an ordinary, recent desktop computer are presented in Table 1. The number of constructions attempted in GRASP was $T = 10,000$. Programs were developed in the Python

---

[3] Instance data and the programs implemented are available in http://www.dcc.fc.up.pt/ jpp/code/occs.

programming language. Although the implementation was careful, there is still room for improvement. Nevertheless, the CPU time required even for large instances is acceptable in practice. The quality of the GRASP solution is in general much superior to that of a purely greedy solution. For the small instances, a visualization of the solutions is provided in Fig. 4.

6. Conclusions

This paper presents RCPP, which are hard combinatorial optimization problems with direct application in the tube industry. Variants of the problem are formulated as mathematical optimization, disjunctive programming model. A greedy method for obtaining heuristic solutions is proposed for the most relevant variant, occurring in a particular industrial setting. This method is extended for including a random component, allowing repeated, semigreedy solution construction, borrowing ideas from GRASP metaheuristics.

Experimental results show that the methods have practical relevance. As this problem is, to the best of our knowledge, presented here for the first time, a set of benchmark instances is suggested for comparing the proposed approach with other methods.

In terms of future research, there are two directions worthy of further investigation: one is tackling more realistic versions of the problem (e.g., considering weights and tubes with sockets); the other is theoretical work on the base model, that is, packing a subset of tubes with maximum value in a single container.

Acknowledgments

This work was partially supported by FEDER, QREN e Compete, within project *OCC: Optimizador de Carga em Contentores, QREN SII&DT Projeto em Co-Promoção 13824*, and includes contributions from its research team. We would like to thank in particular Eng. Jorge Leite, from Fersil, Portugal, for his valuable insights. We would also like to thank two anonymous reviewers for their constructive comments on a previous version of this paper.

Partial support from project "NORTE-07-0124-FEDER-000057," under the North Portugal Regional Operational Programme (ON.2 O Novo Norte) and the National Strategic Reference Framework through the European Regional Development Fund, and by national funds through Fundação para a Ciência e a Tecnologia (FCT) is also acknowledged.

References

Birgin, E.G., Bustamante, L.H., Callisaya, H.F., Martínez, J.M., 2013. Packing circles within ellipses. *International Transactions in Operational Research* 20, 3, 365–389.

Birgin, E.G., Gentil, J.M., 2010. New and improved results for packing identical unitary radius circles within triangles, rectangles and strips. *Computers & Operations Research* 37, 7, 1318–1327.

Birgin, E.G., Martínez, J., Ronconi, D., 2005. Optimizing the packing of cylinders into a rectangular container: a nonlinear approach. *European Journal of Operational Research* 160, 1, 19–33.

Birgin, E.G., Sobral, F.N.C., 2008. Minimizing the object dimensions in circle and sphere packing problems. *Computers and Operations Research* 35, 7, 2357–2375.

Castillo, I., Kampas, F.J., Pintér, J.D., 2008. Solving circle packing problems by global optimization: numerical results and industrial applications. *European Journal of Operational Research* 191, 3, 786–802.

Feo, T., Resende, M., 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6, 109–133.

George, J.A., George, J.M., Lamar, B.W., 1995. Packing different-sized circles into a rectangular container. *European Journal of Operational Research* 84, 3, 693–712.

Hifi, M., M'Hallah, R., 2009. A literature review on circle and sphere packing problems: models and methodologies. *Advances in Operations Research* 2009, 4, 1–22.

Hifi, M., Paschos, V.T., Zissimopoulos, V., 2004. A simulated annealing approach for the circular cutting problem. *European Journal of Operational Research* 159, 2, 430–448.

Huang, W., Li, Y., Akeb, H., Li, C., 2004. Greedy algorithms for packing unequal circles into a rectangular container. *Journal of the Operational Research Society* 56, 5, 539–548.

Huang, W., Li, Y., Li, C.M., Xu, R.C., 2006. New heuristics for packing unequal circles into a circular container. *Computers and Operations Research* 33, 8, 2125–2142.

Kallrath, J., 2009. Cutting circles and polygons from area-minimizing rectangles. *Journal of Global Optimization* 43, 2–3, 299–328.

Kubach, T., Bortfeldt, A., Gehring, H., 2009. Parallel greedy algorithms for packing unequal circles into a strip or a rectangle. *Central European Journal of Operations Research* 17, 4, 461–477.

Lenstra, J., Rinnooy Kan, A., 1979. Complexity of packing, covering, and partitioning problems. In Schrijver, A. (ed.) *Packing and Covering in Combinatorics*. Mathematisch Centrum, Amsterdam, pp. 275–291.

Stephenson, K., 2003. Circle packing: a mathematical tale. *Notices of the American Mathematical Society* 50, 11, 1376–1388.

Stoyan, Y.G., Yaskov, G., 2004. A mathematical model and a solution method for the problem of placing various-sized circles into a strip. *European Journal of Operational Research* 156, 3, 590–600.

Stoyan, Y.G., Yaskov, G., 2012. Packing equal circles into a circle with circular prohibited areas. *International Journal of Computer Mathematics* 89, 10, 1355–1369.

Stoyan, Y.G., Yaskov, G., 2014. Packing unequal circles into a strip of minimal length with a jump algorithm. *Optimization Letters* 8, 3, 949–970.

Stoyan, Y.G., Zlotnik, M.V., Chugay, A.M., 2012. Solving an optimization packing problem of circles and non-convex polygons with rotations into a multiply connected region. *Journal of the Operational Research Society* 63, 3, 379–391.

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