View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Open Research Exeter

Zuse Institute Berlin

Takustrasse 7
D-14195 Berlin-Dahlem
Germany

AMBROS GLEIXNER
STEPHEN J. MAHER
BENJAMIN MÜLLER
JOÃO PEDRO PEDROSO

Exact Methods for Recursive Circle Packing

This report has been published in Annals of Operations Research. Please cite as:

Gleixner et al., Price-and-verify: a new algorithm for recursive circle packing using Dantzig–Wolfe decomposition. Annals of Operations Research, 2018, DOI:10.1007/s10479-018-3115-5

ZIB Report 17-07 (January 2018)

Zuse Institute Berlin
Takustrasse 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Exact Methods for Recursive Circle Packing

Ambros Gleixner* Stephen J. Maher* Benjamin Müller*
João Pedro Pedroso†

January 2, 2019

Abstract

Packing rings into a minimum number of rectangles is an optimization problem which appears naturally in the logistics operations of the tube industry. It encompasses two major difficulties, namely the positioning of rings in rectangles and the recursive packing of rings into other rings. This problem is known as the Recursive Circle Packing Problem (RCPP). We present the first dedicated method for solving RCPP that provides strong dual bounds based on an exact Dantzig-Wolfe reformulation of a nonconvex mixed-integer nonlinear programming formulation. The key idea of this reformulation is to break symmetry on each recursion level by enumerating one-level packings, i.e., packings of circles into other circles, and by dynamically generating packings of circles into rectangles. We use column generation techniques to design a "price-and-verify" algorithm that solves this reformulation to global optimality. Extensive computational experiments on a large test set show that our method not only computes tight dual bounds, but often produces primal solutions better than those computed by heuristics from the literature.

1 Introduction

Packing problems appear naturally in a wide range of real-world applications. They contain two major difficulties. First, complex geometric objects need to be selected, grouped, and packed into other objects, e.g., warehouses, containers, or parcels. Second, these objects need to be packed in a dense, non-overlapping way and, depending on the application, respect some physical constraints. Modeling packing problems mathematically often leads to nonconvex mixed-integer nonlinear programs (MINLPs). Solving them to global optimality is a challenging task for state-of-the-art MINLP solvers. In this paper, we will study exact algorithms for solving a particularly complex version involving the *recursive* packing of 2-dimensional rings. This variant has real-world applications in the tube industry.

The Recursive Circle Packing Problem (RCPP) has been introduced recently by Pedroso et al. [25]. The objective of RCPP is to select a minimum number of rectangles of the same size such that a given set of rings can be packed into these rectangles in a non-overlapping way. A ring is characterized by an internal and an external radius. Rings can be put recursively into larger ones or directly into a rectangle. A set of rings packed into a rectangle is called a *feasible packing* if and only if all rings lie within the boundary of the rectangle and do not intersect each other. Figure 1 gives two examples of a feasible packing.

*Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, {benjamin.mueller,gleixner,maher}@zib.de
†Faculdade de Ciências, Universidade do Porto, Rua do Campo Alegre, 4169-007 Porto, Portugal, jpp@fc.up.pt

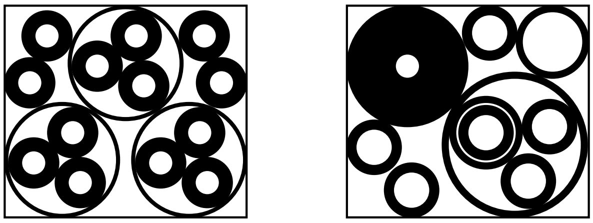

Figure 1: Two feasible packings of rings into rectangles.

Pedroso et al. [25] present a nonconvex MINLP formulation for RCPP. The key idea of the developed MINLP is to use binary variables in order to indicate whether a ring is packed inside another larger ring or directly into a rectangle. Due to a large number of binary variables and nonconvex quadratic constraints, the model is not of practical relevance and can only be used to solve very small instances. However, the authors present a well-performing local search heuristic to find good feasible solutions.

The purpose of this article is to present the first exact column generation algorithm for RCPP based on a Dantzig-Wolfe decomposition, for which, so far, only heuristics exist. The new method is able to solve small- and medium-size instances to global optimality and computes good feasible solutions and tight dual bounds for larger ones. We first develop a reformulation, which is similar to the classical reformulation for the Cutting Stock Problem [14], however, featuring nonlinear and nonconvex sub-problems. This formulation breaks the symmetry between equivalent rectangles. As a second step, we combine this reformulation with an enumeration scheme for patterns that are characterized by rings packed inside other rings. Such patterns only allow for a one-level recursion and break the symmetry between rings with the same internal and external radius in each rectangle. Finally, we present how to solve the resulting reformulation with a column generation algorithm.

Finally, we present how to solve the resulting reformulation with a specialized column generation algorithm that we call *price-and-verify*. The algorithm does not only generate new patterns but also verifies patterns that are found during the enumeration scheme dynamically, i.e., only when required for solving the continuous relaxation of the reformulation.

2 Background

The problem of finding dense packings of geometric objects has a rich history that goes back to Kepler's Conjecture in 1611, which has been proven recently by Hales et al. [16]. Packing identical or diverse objects into different geometries like circles, rectangles, and polygons remains a relevant topic and has been the focus of much research during the last decades. The survey by Hifi and M'Hallah [17] reviews the most relevant results for packing 2- and 3-dimensional spheres into regions in the Euclidean space. Applications, heuristics, and exact strategies of packing arbitrarily sized circles into a container are presented by Castillo et al. [4]. Costa et al. [7] use a spatial branch-and-bound algorithm to find good feasible solutions for the case of packing identical circles. They propose different symmetry-breaking constraints to tighten the convex relaxation and improve the success rate of local nonlinear programming algorithms.

Closely related to packing problems are plate-cutting problems, in which convex objects, e.g., circles, rectangles, or polygons, are cut from different convex geometries [9]. Modeling these problems leads typically to nonlinear programs, for which exact mathematical programming solutions are described by Kallrath [20]. Minimizing the volume of a box for an overlap-free placement of ellipsoids was studied by Kallrath [21]. His closed, non-convex nonlinear programming formulation uses the entries of the rotation matrix as variables and can be used to get feasible solutions for instances with up to 100 ellipsoids.

The general type of the problem finds application in the tube industry, where shipping costs represent a large fraction of the total costs of product delivery. Tubes will be cut to the same length of the container in which they may be shipped. In order to reduce idle space inside containers, smaller tubes might be placed inside larger ones. Packing tubes as densely as possible reduces the total number of containers needed to deliver all tubes and thus has a large impact on the total cost.

RCPP is a generalization of the well-known, $\mathcal{NP}$-hard [23, ?] Circle Packing Problem (CPP) and therefore is also $\mathcal{NP}$-hard. Reducing an instance of CPP to RCPP can be done by setting all internal radii to zero, i.e., by forbidding to pack rings into larger rings. Typically, problems like RCPP contain multiple sources of symmetry. Any permutation of rectangles, i.e., relabeling rectangles, constitutes an equivalent solution to RCPP. Even worse, there is also considerable symmetry inside a rectangle. First, rotating or reflecting a rectangle gives an equivalent rectangle since both contain the same set of rings. Second, two rings with same internal and external radius can be exchanged arbitrarily inside a rectangle, again resulting in an equivalent rectangle packing.

One possible way to break the symmetry of RCPP is to add symmetry-breaking constraints, which have been frequently used for scheduling problems, e.g., lot-sizing problems [19]. An alternative approach is the use of decomposition techniques. These techniques aggregate identical sub-problems in order to reduce symmetry and typically strengthen the relaxation of the initial formulation. A well-known decomposition technique is the Dantzig-Wolfe decomposition [13]. It is an algorithm to decompose Linear Programs (LPs) into a master and, in general, several sub-problems. Column generation is used with this decomposition to improve the solvability of large-scale linear programs. Embedded in a branch-and-bound algorithm, it can be used to solve mixed-integer linear programs (MILPs), e.g., bin packing [33], two dimensional bin packing [26], cutting stock [14, 15], multi-stage cutting stock [?], and many other problems. Fortz et al. [11] applied a Dantzig-Wolfe decomposition to a stochastic network design problem with a convex nonlinear objective function. While most implementations of Dantzig-Wolfe are problem-specific, a number of frameworks have been implemented for automatically applying a Dantzig-Wolfe decomposition to a compact MILP formulation, see [3, 12, 27].

In this paper we apply a Dantzig-Wolfe decomposition to an MINLP formulation of RCPP and present the first exact solution method that is able to solve practically relevant instances. The rest of the paper is organized as follows. In Section 3, we introduce basic notation and discuss the limitations of a compact MINLP formulation for RCPP. Section 4 presents a first Dantzig-Wolfe decomposition of the MINLP formulation. After introducing the concept of circular and rectangular patterns, we extend the formulation from Section 4 in Section 5 to our final formulation for RCPP. Afterwards, we present a column generation algorithm to solve this formulation, which uses an enumeration scheme introduced in Section 6.1. Section 6.2 shows how to prove valid dual and primal bounds, even when difficult sub-problems cannot be solved to optimality. Finally, in Section 7, we analyze the performance of our method on a large test set containing 800 synthetic instances and nine real-world instances from the tube industry. Section 8 gives concluding remarks.

3 Problem Statement

Consider a set $\mathcal{T} := \{1, \dots, T\}$ for T different types of rings and an infinite number of rectangles. In what follows, we consider each rectangle to be of size $W \in \mathbb{R}_+$ times $H \in \mathbb{R}_+$ and to be placed in the Euclidean plane such that the corners are $(0,0)$, $(0,W)$, $(H,0)$, and (W,H).

For each ring type $t \in \mathcal{T}$ we are given an internal radius $r_t \in \mathbb{R}_+$ and an external radius $R_t \in \mathbb{R}_+$ such that

$$r_t \leq R_t \leq \min\{W, H\}.$$

Also, each ring type $t \in \mathcal{T}$ has a demand $D_t \in \mathbb{Z}_+$. We assume without loss of generality that $\mathcal{T}$ is sorted such that $R_1 \leq \ldots \leq R_T$. To simplify the notation, we denote by $n := \sum_{t \in \mathcal{T}} D_t$ the total number of individual rings and denote by $\mathcal{R} := \{1, \ldots, n\}$ the corresponding index set. The function $\tau : \mathcal{R} \to \mathcal{T}$ maps each individual ring to its corresponding type. By a slight abuse of notation, we identify with r_i and R_i the internal and external radius of ring $i \in \mathcal{R}$, i.e., $R_i = R_{\tau(i)}$ and $r_i = r_{\tau(i)}$.

The task in RCPP is to pack all rings in $\mathcal{R}$ into the smallest number of rectangles. Rings must lie within the boundary of a rectangle and must not intersect each other. More precisely, a feasible solution to RCPP can be encoded as a 3-tuple $(c, x, y) \in \{1, \ldots, k\}^n \times \mathbb{R}^n \times \mathbb{R}^n$ where (x_i, y_i) denotes the center of ring $i \in \mathcal{R}$ inside rectangle $c_i \in \{1, \ldots, k\}$, and an upper bound on the number of rectangles needed is given by $k \leq n$. The number of used rectangles is equal to the cardinality of $\{c_1, \ldots, c_n\}$, i.e., the number of distinct integer values. Rings must not intersect the boundary of the rectangle, i.e.,

$$R_i \leq x_i \leq W - R_i \text{ for all } i \in \mathcal{R}, \tag{1}$$

$$R_i \leq y_i \leq H - R_i \text{ for all } i \in \mathcal{R}. \tag{2}$$

For a given 3-tuple (c, x, y) we denote by

$$A(i) := \left\{ (\tilde{x}, \tilde{y}) \in \mathbb{R}^2 \mid r_i < \left\| \begin{pmatrix} \tilde{x} - x_i \\ \tilde{y} - y_i \end{pmatrix} \right\|_2 < R_i \right\}$$

the area occupied by ring i in rectangle c_i. The non-overlapping condition between different rings is equivalent to

$$A(i) \cap A(j) = \emptyset \tag{3}$$

for all $i \neq j$ with $c_i = c_j$.

RCPP can be equivalently formulated as an MINLP, see Pedroso et al. [25]. The formulation consists of four different types of variables:

- $(x_i, y_i) \in \mathbb{R}^2$, the center of ring $i \in \mathcal{R}$,
- $z_c \in \{0, 1\}$, a decision variable whether rectangle $c \in \{1, \ldots, k\}$ is used,
- $w_{i,c} \in \{0, 1\}$, a decision variable whether ring i is directly placed in rectangle c, and
- $u_{i,j} \in \{0, 1\}$, a decision variable whether ring i is directly placed in ring j.

We say that a ring $i \in \mathcal{R}$ is *directly placed* in another ring j or rectangle c if it is not contained in another, larger ring inside j or c, respectively. Condition (3) can be modeled by the constraints

$$\left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\|_2^2 \geq (R_i + R_j)^2 (w_{i,c} + w_{j,c} - 1) \qquad \text{for all } i, j \in \mathcal{R} : i \neq j,\ c \in \{1, \ldots, k\}, \tag{4}$$

$$\left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\|_2^2 \geq (R_i + R_j)^2 (u_{i,h} + u_{j,h} - 1) \qquad \text{for all } i, j, h \in \mathcal{R} : i \neq j \wedge i \neq h \wedge j \neq h, \tag{5}$$

$$\left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\|_2 \leq r_j - R_i + M_{i,j}(1 - u_{i,j}) \qquad \text{for all } i, j \in \mathcal{R} : R_i \leq r_j, \tag{6}$$

which are nonconvex nonlinear inequality constraints. Inequality (4) guarantees that no two rings $i, j \in \mathcal{R}$ overlap when they are directly placed inside the same rectangle c, i.e., if $w_{i,c} = w_{j,c} = 1$. Similarly, inequality (5) ensures that no two rings intersect inside another ring. Constraint (6) ensures that if $u_{i,j} = 1$, then ring i is directly placed inside j and does not intersect its boundary. Otherwise, the constraint is disabled. An appropriate value for $M_{i,j}$ to guarantee that the conditions of (6) are satisfied is

$$M_{i,j} := \sqrt{(W - R_i - R_j)^2 + (H - R_i - R_j)^2},$$

which is the maximum distance between two rings $i, j \in \mathcal{R}$ in a W times H rectangle that do not overlap.

The full MINLP model for RCPP reads as follows:

$$\begin{aligned}
\min \quad & \sum_{c=1}^{k} z_c && && \text{(7a)}\\
\text{s.t.} \quad & w_{i,c} \le z_c && \text{for all } i \in \mathcal{R},\ c \in \{1,\ldots,k\} && \text{(7b)}\\
& \sum_{c=1}^{k} w_{i,c} + \sum_{j \in \mathcal{R}} u_{i,j} = 1 && \text{for all } i \in \mathcal{R} && \text{(7c)}\\
& (1), (2), (4), (5), (6) && &&\\
& (x_i, y_i) \in \mathbb{R}^2 && \text{for all } i \in \mathcal{R} &&\\
& z_c \in \{0,1\} && \text{for all } c \in \{1,\ldots,k\} &&\\
& w_{i,c} \in \{0,1\} && \text{for all } i \in \mathcal{R},\ c \in \{1,\ldots,k\} &&\\
& u_{i,j} \in \{0,1\} && \text{for all } i, j \in \mathcal{R} : i \neq j &&
\end{aligned}$$

Finally, the objective function (7a) minimizes the total number of rectangles used. Constraint (7b) guarantees that we can pack a ring inside a rectangle if and only if the rectangle is used. Each ring needs to be packed into another ring or a rectangle, which is ensured by (7c).

Remark 1. Formulation (7) can be adapted for maximizing the load of a single rectangle. Roughly speaking, we need to fix $z_c = 0$ for each $c > 1$ and replace each variable $w_{i,c}$ by w_i, which then turns into an indicator whether ring $i \in \mathcal{R}$ has been packed or not. The objective function (7a) changes to

$$\max \sum_{i \in \mathcal{R}} \alpha_i w_i,$$

where $\alpha_i \in \mathbb{R}_+$ is a non-negative weight for each ring $i \in \mathcal{R}$.

Unfortunately, general MINLP solvers require a considerable amount of time to solve Formulation (7) because it contains a large number of variables, many nonconvex constraints, and much symmetry. Even the smallest instance of our test set, containing 54 individual rings, could not be solved within several hours by SCIP 3.2.1 [29] nor by BARON 16.8.24 [22, 28]. A common method used to improve the solvability of geometric packing problems is to add symmetry breaking constraints. However, strengthening (7) by adding the constraints

$$\begin{aligned}
z_c &\ge z_{c+1} && \text{for all } c \in \{1,\ldots,k-1\}, && (8)\\
u_{i,j} &\ge u_{h,j} && \text{for all } j \in \mathcal{R},\ i < h : \tau(i) = \tau(h), \text{ and} && (9)\\
w_{i,c} &\ge w_{j,c} && \text{for all } c \in \{1,\ldots,k\},\ i < j : \tau(i) = \tau(j), && (10)
\end{aligned}$$

which break symmetry by ordering rectangles and rings of the same type, had no impact on the solvability of RCPP.

The reason for the poor performance is that (7) contains $O(kn^2+n^3)$ many difficult nonconvex constraints of the form (4) and (5). Even worse, the binary variables in those constraints appear with a big-M, which is known to result in weak linear programming relaxations. Another difficulty is the symmetry in the model that is not eliminated by (8), (9), or (10). Each reflection or rotation of a feasible packing of a rectangle and rings inside a ring yields another, equivalent, solution.

In the following, we present two formulations based on a Dantzig-Wolfe decomposition to break the remaining symmetry in (7). The first formulation breaks the symmetry between rectangles. The second is an extension of the first one and additionally breaks symmetry between rings of the same type inside the rectangles and other rings.

To overcome the difficulty of the exponential number of variables in the Dantzig-Wolfe decompositions, we use column generation to solve the continuous relaxations of both formulations. The pricing problem of the first formulation is the maximization version of (7). The pricing problem of the second formulation is a maximization version of the CPP.

4 Cutting Stock Reformulation

Dantzig-Wolfe decomposition [13] is a classic solution approach for structured LPs. It can be used to decompose an MILP into a *master problem* and one or several *sub-problems.* Advantages of the decomposition are that it yields stronger relaxations if the sub-problems are nonconvex and can aggregate equivalent sub-problems to reduce symmetries [24, 8].

In this section, we apply a Dantzig-Wolfe decomposition to (7) and obtain a reformulation that is similar to the one of the one-dimensional Cutting Stock Problem [30]. The key idea is to reformulate RCPP in order to not assign rings explicitly to rectangles, but rather choose a set of feasible rectangle packings to cover the demand for each ring type. The resulting reformulation is a pure integer program (IP) containing exponentially many variables. We solve this reformulation via column generation.

We call a vector $F \in \mathbb{Z}_+^T$ *packable* if it is possible to pack F_t many rings for all types $t \in \mathcal{T}$ together (and thus a total number of $\sum_{t\in\mathcal{T}} F_t$ many rings) into a $W \times H$ rectangle. A packable $F \in \mathbb{Z}_+^T$ corresponds to a feasible solution of (7) when considering only a single rectangle. Denote by

$$\mathcal{F} := \{F \in \mathbb{Z}_+^T \,:\, F \text{ is packable}\}$$

the set of all feasible packings of rings into a rectangle. Note that without loss of generality, we can bound F_t by the demand D_t for each $t \in \mathcal{T}$. After applying Dantzig-Wolfe decomposition to (7), we obtain the following formulation, $DW(\mathcal{F})$:

$$\min \sum_{F\in\mathcal{F}} z_F \tag{11a}$$

$$\text{s.t.} \quad \sum_{F\in\mathcal{F}} F_t \cdot z_F \geq D_t \qquad \text{for all } t \in \mathcal{T} \tag{11b}$$

$$z_F \in \mathbb{Z}_+ \qquad \text{for all } F \in \mathcal{F} \tag{11c}$$

This formulation contains an exponential number of integer variables with respect to T. Each variable z_F counts how often F is chosen. Minimizing the total number of rectangles is equivalent to (11a). Inequality (11b) ensures that each ring of type t is packed at least D_t many times. In the following, we call the LP relaxation of $DW(\mathcal{F}')$ for $\mathcal{F}' \subseteq \mathcal{F}$ the *restricted master problem* of $DW(\mathcal{F})$.

An advantage of (11) is that we do not explicitly assign rings to positions inside some rectangles, like in (7), nor distinguish between rings of the same type. This breaks symmetry that arises from simply permuting rectangles. However, $\mathcal{F}$ is a priori unknown and its size is exponential in the input size of an RCPP instance.

A column generation algorithm is used to solve the LP relaxation of $DW(\mathcal{F})$ by iteratively updating $DW(\mathcal{F})$ with improving columns corresponding to feasible rectangular packings. A feasible rectangular packing $F \in \mathcal{F}$ is an improving column if the corresponding constraint

$$\sum_{t \in \mathcal{T}} F_t \pi_t^* \leq 1$$

is violated by the dual solution π^* of the restricted master problem $DW(\mathcal{F}')$. The LP relaxation of $DW(\mathcal{F})$ is solved to optimality when $\sum_{t \in \mathcal{T}} F_t \pi_t^* \leq 1$ holds for all $F \in \mathcal{F} \setminus \mathcal{F}'$. Otherwise, an $F \in \mathcal{F} \setminus \mathcal{F}'$ with $\sum_{t \in \mathcal{T}} F_t \pi_t^* > 1$ is added to $\mathcal{F}'$ and the process iterates until $DW(\mathcal{F})$ is solved to optimality.

The most violated dual constraint is found by solving

$$\min \left\{ 1 - \sum_{t \in \mathcal{T}} \pi_t^* F_t : F \in \mathbb{Z}_+^T,\ F \text{ is packable} \right\}. \tag{12}$$

This problem is a maximization variant of RCPP for which we need to find a subset of rings such that they can be packed into a single rectangle, see Remark 1. The objective gain of each ring of type t is exactly π_t^*. The LP relaxation of (11) is solved to optimality if the solution value of (12) is non-negative. Otherwise, we find an improving column that, after it has been added, might decrease the objective value of the restricted master problem $DW(\mathcal{F}')$.

As explained in Remark 1, (12) can be modeled as a nonconvex MINLP that contains the selection and positioning of rings in a rectangle. Unfortunately, solving this problem is very difficult for a general MINLP solver. In our experiments none of the resulting pricing problems (12) could be solved to optimality or even generate an improving column for $DW(\mathcal{F}')$ in two hours. For this reason, Formulation (11) is not suitable for solving RCPP to global optimality.

5 Pattern-based Dantzig-Wolfe Decomposition using One-level Packings

The main drawback of (11) is that the resulting pricing problems (12) are intractable. This is due to two major difficulties, i) the positioning of rings inside a rectangle and ii) the combinatorial decisions of how to put rings into other rings. Together, they make the sub-problems much more difficult to solve than the IP master problem. The recursive decisions of packing rings into each other introduces much symmetry to (12). Rings of the same type, and all rings packed inside those, can be swapped inside a rectangle and yield an equivalent solution. This symmetry appears in each recursion level of the sub-problems. See Figure 2 for an example of this kind of symmetry.

The main idea of the following reformulation is to break this type of symmetry and shift the recursive decisions from the sub-problem to the master problem. This helps balance the complexity of both problems, which is crucial when using a column generation algorithm.

In the following, we introduce the concept of *circular* and *rectangular patterns*. These patterns describe possible packings of circles into rings or rectangles. The circles act like placeholders.

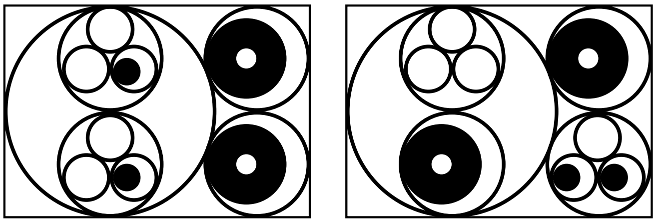

Figure 2: Two equivalent feasible packings obtained by swapping rings packed inside other rings.

Specifically, the circles just describe what type of rings might be placed in the circles, but not how these rings are filled with other rings. After choosing one pattern we are able to choose other patterns that fit into the circles of the selected one. The recursive part of RCPP boils down to a counting problem of patterns and minimizing the total number of rectangles.

5.1 Circular Patterns

A *circle* of type $t \in \mathcal{T}$ is a ring with external radius R_t and inner radius $r_t = 0$. This means that neither circles nor rings can be put into a circle. Similarly to the definition of elements in $\mathcal{F}$, we call a tuple $(t, P) \in \mathcal{T} \times \mathbb{Z}_+^T$ a *circular pattern* if it is possible to pack P_1 many circles of type 1, P_2 many circles of type 2, ..., P_T many circles of type T into a larger ring of type t. As an example, $(3, (2, 1, 0))$ is a circular pattern with an outer ring of type three which contains two circles with external radius R_1 and one circle with radius R_2. Since R_3 can not be pack within any ring, the final index will always be 0. However, it is included in the definition of circular patterns for completeness. Figure 3 shows all possible packings for three different ring types.

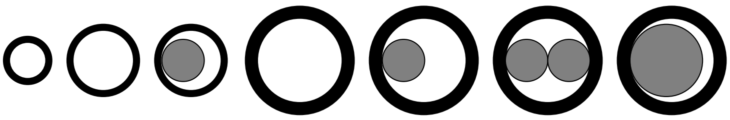

Figure 3: All possible circular patterns for three different ring types:
$(1, (0, 0, 0)), (2, (0, 0, 0)), (2, (1, 0, 0)), (3, (0, 0, 0)), (3, (1, 0, 0)), (3, (2, 0, 0)), (3, (0, 1, 0))$

Using the definition of circular patterns we decompose a packing of rings into a ring R. Each ring that is directly placed into R, i.e., is not contained in another larger ring, is replaced by a circle with the same external radius. We apply the decomposition to each replaced ring recursively. As a result, ring R decomposes into a set of circular patterns. Starting from these circular patterns, we can reconstruct R by recursively replacing circles in a pattern with other circular patterns. We denote by

$$\mathcal{CP} := \{(t, P) \in \mathcal{T} \times \mathbb{Z}_+^T \mid (t, P) \text{ is a circular pattern}\}$$

the set of all possible circular patterns. The previously described decomposition shows that any recursive packing of rings can be constructed by using circular patterns of $\mathcal{CP}$.

In general, the cardinality of $\mathcal{CP}$ is exponential in T and depends on the internal and external radii of the rings. For example, increasing the inner radius of the largest ring will lead to many more possibilities to pack circles into this ring. In contrast, decreasing external radii results in fewer circular patterns.

Figure 3 shows that there are circular patterns that are dominated by others, e.g., the third pattern dominates second one. Using dominated circular patterns would leave some unnecessary free space in some rectangles, which can be avoided in an optimal solution. In Section 6.1 we discuss this domination relation and present an algorithm to compute all non-dominated circular patterns.

5.2 Rectangular Patterns

In the following reformulation of RCPP, we use circular patterns to shift the decisions of how to put rings into other rings to the master problem. Similar to a circular pattern, we call $P \in \mathbb{Z}_+^{\mathcal{T}}$ a *rectangular pattern* if and only if P_t many circles with radius R_t for each $t \in \mathcal{T}$ can be packed together into a rectangle of size W times H. Let

$$\mathcal{RP} := \{P \in \mathbb{Z}_+^T \mid P \text{ is a rectangular pattern}\}$$

be the set of all rectangular patterns. As in Formulation (11), only the number of packed circles matters, not their position in the rectangular pattern.

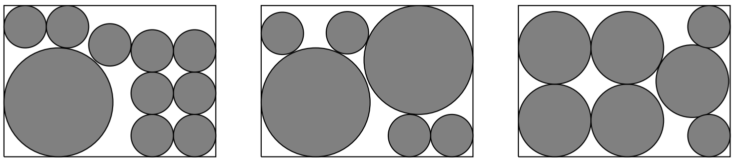

Figure 4: Three different rectangular patterns:
$(9, 0, 1), (4, 0, 2), (2, 5, 0)$

In contrast to verifying if $(t, P) \in \mathcal{T} \times \mathbb{Z}_+^T$ is a circular pattern, checking whether $P \in \mathbb{Z}_+^T$ is a rectangular pattern—a classical CPP—can be much more difficult. Typically, many more circles fit into a rectangle than into a ring. This results in a large number of circles that need to be considered in a verification problem, which is in practice difficult to solve.

5.3 Exploiting Recursion

Using the circle and rectangle patterns described above, we develop a pattern-based Dantzig-Wolfe decomposition for RCPP with nonlinear sub-problems. Instead of placing rings explicitly into each other, we use patterns to remodel the recursive part. The key idea is that circles, inside rectangular or circular patterns, are replaced by circular patterns with the same external radius. These circular patterns contain circles that can be replaced by other circular patterns. More precisely, after choosing a rectangular pattern $P \in \mathcal{RP}$, it is possible to choose P_t circular patterns of the form $(t, P') \in \mathcal{CP}$, which can be placed into P. Again, for each P' we can choose $P'_{t'}$ many circular patterns of the form (t', P''), which can be placed in P'. This process can continue until the smallest packable circle is considered. The recursive structure of the RCPP—the placement of rings into other rings—is modeled by counting the number of used rectangular and circular patterns.

Figure 5 illustrates this idea. A circular pattern replaces a circle if there is an arrow from the pattern to the circle. Each circle of a pattern can be used as often as the pattern is used. It follows that the number of outgoing edges of a circular pattern is equal to the number of uses of the pattern. The combinatorial part of RCPP reduces to adding edges from circular patterns to circles.

Following this idea, we introduce integer variables

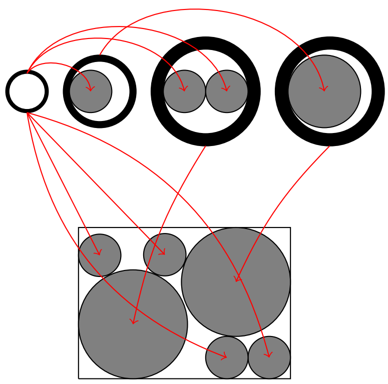

Figure 5: An example with four circular patterns and a rectangular pattern showing how patterns are used to model the combinatorial part of RCPP. Each line connects a circular pattern to a circle. The number of outgoing edges is equal to the number of rings that are used.

- $z_C \in \mathbb{Z}_+$ for each circular pattern $C \in \mathcal{CP}$ and
- $z_P \in \mathbb{Z}_+$ for each rectangular $P \in \mathcal{RP}$

in order to count the number of used circular and rectangular patterns (in the pattern-based formulation). We reformulate RCPP by the following IP formulation $PDW(\mathcal{RP})$, which is similar to the multi-stage cutting stock formulation that has been presented by Muter et al. [?].

$$\min \sum_{P \in \mathcal{RP}} z_P \tag{13a}$$

$$\text{s.t.} \sum_{C=(t,P) \in \mathcal{CP}} z_C \geq D_t \qquad \text{for all } t \in \mathcal{T} \tag{13b}$$

$$\sum_{P \in \mathcal{RP}} P_t \cdot z_P + \sum_{C=(t',P) \in \mathcal{CP}} P_t \cdot z_C \geq \sum_{C=(t,P) \in \mathcal{CP}} z_C \qquad \text{for all } t \in \mathcal{T} \tag{13c}$$

$$z_C \in \mathbb{Z}_+ \qquad \text{for all } C \in \mathcal{CP} \tag{13d}$$

$$z_P \in \mathbb{Z}_+ \qquad \text{for all } P \in \mathcal{RP} \tag{13e}$$

Objective (13a) minimizes the total number of used rectangles. Constraint (13b) ensures that the demand for each ring type is satisfied. The recursive decisions how to place rings into each other are implicitly modeled by (13c). Each selection of a pattern $P \in \mathcal{RP}$ or $(t', P) \in \mathcal{CP}$ allows us to choose P_t circular patterns of the type t. Note that at least one rectangular pattern needs to be selected before circular patterns can be packed. This is true because the largest ring only fits into a rectangular pattern. To aid in understanding the formulation of $PDW(\mathcal{RP})$, a small example based on the circular and rectangular patterns in Figures 3 and 4 respectively is presented in Example 1.

Example 1. Let $\{C_1, \ldots, C_7\}$ be the set of circular patterns of Figure 3 and $\{P_1, P_2, P_3\}$ be the subset of rectangular patterns of Figure 4. The patterns are labeled from left to right.

Problem (13) then reads as

$$\min z_{P_1} + z_{P_2} + z_{P_3} \tag{14a}$$

$$\text{s.t.} \quad z_{C_1} \geq D_1 \tag{14b}$$

$$z_{C_2} + z_{C_3} \geq D_2 \tag{14c}$$

$$z_{C_4} + z_{C_5} + z_{C_6} + z_{C_7} \geq D_3 \tag{14d}$$

$$z_{C_3} + z_{C_5} + 2z_{C_6} + 9z_{P_1} + 4z_{P_2} + 2z_{P_3} \geq z_{C_1} \tag{14e}$$

$$z_{C_7} + 5z_{P_3} \geq z_{C_2} + z_{C_3} \tag{14f}$$

$$z_{P_1} + 2z_{P_2} \geq z_{C_4} + z_{C_5} + z_{C_6} + z_{C_7} \tag{14g}$$

$$z_{C_i} \in \mathbb{Z}_+ \quad \text{for all } i \in \{1, \dots, 7\} \tag{14h}$$

$$z_{P_i} \in \mathbb{Z}_+ \quad \text{for all } i \in \{1, 2, 3\} \tag{14i}$$

Constraints (14b)–(14d) ensure that the demand for each ring type is satisfied. The left-hand side of constraints (14b)–(14d) only contain columns corresponding to the circular patterns of ring type 1 to 3 respectively. The constraints (14e)–(14g) model the recursive structure of the problem. Columns corresponding to circular patterns for ring types 1 to 3 are observed on the right-hand side of constraints (14e)–(14g) respectively. The left-hand side of constraints (14e)–(14g) contain columns corresponding to rectangular and circular patterns that pack the ring type represented on the right-hand side of the respective constraints.

A drawback of (13) is the exponential number of rectangular and circular pattern variables. To address this difficulty, we develop a column enumeration algorithm to compute all (relevant) circular patterns used in (13), which is presented in Section 6.1. We observed in our experiments that for many instances this algorithm successfully enumerates all circular patterns in a reasonable amount of time.

Since the size of the rectangles is much larger than the external radii R, it is intractable to enumerate all rectangular patterns. To overcome this difficulty, we use a column generation approach to solve the LP relaxation of (13) that dynamically generates rectangular patterns variables. We call the LP relaxation of $PDW(\mathcal{RP}')$ the restricted master problem of (13) for a subset of rectangular patterns $\mathcal{RP}' \subseteq \mathcal{RP}$. In order to find an improving column for $PDW(\mathcal{RP}')$, we solve a weighted CPP for a single rectangle.

More precisely, let $\lambda \in \mathbb{R}^T_+$ be the non-negative vector of dual multipliers for Constraints (13c) after solving the LP relaxation of $PDW(\mathcal{RP}')$ for the current set of rectangular patterns $\mathcal{RP}' \subset \mathcal{RP}$. To compute a rectangular pattern with negative reduced cost we solve

$$\min_{P \in \mathcal{RP} \setminus \mathcal{RP}'} \left\{ 1 - \sum_{t \in \mathcal{T}} \lambda_t P_t \right\}, \tag{15}$$

which can be modeled as a weighted CPP for a single rectangle. Let P^* be an optimal solution to (15). If $1 - \sum_{t \in \mathcal{T}} \lambda_t P^*_t$ is negative, then P^* is an improving rectangular pattern, whose corresponding variable needs to be added to the restricted master problem of $PDW(\mathcal{RP}')$. Otherwise, the LP relaxation of (13) is solved to optimality.

The pricing problem is $\mathcal{NP}$-hard [23] and difficult to solve in practice. The number of variables in this problem depends on the number of different ring types T and on the demand vector D. When solving (15) we need to consider the index set of individual circles

$$\{i^1_1, \dots, i^1_{D_1}, i^2_1, \dots, i^2_{D_2}, \dots, i^T_1, \dots, i^T_{D_T}\}$$

containing D_t indices that correspond to circles with radius R_t for each $t \in \mathcal{T}$. The number of copies for type t can be reduced to $\min\{D_t, \left\lfloor \frac{\pi(R_t)^2}{WH} \right\rfloor\}$, which is an upper bound on the number of rings of type t in a W times H rectangle. For simplicity, denote with $\mathcal{C}$ the index set of all individual circles. Let R_i be the external radius and $\tau(i)$ the type of circle $i \in \mathcal{C}$. The circle packing problem formulation of (15) reads then as

$$\min 1 - \sum_{i \in \mathcal{C}} \lambda_{\tau(i)} z_i \tag{16a}$$

$$\text{s.t.} \quad \left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\|_2^2 \geq (R_i + R_j)^2 (z_i + z_j - 1) \qquad \text{for all } i, j \in \mathcal{C} : i \neq j \tag{16b}$$

$$R_i \leq x_i \leq W - R_i \qquad \text{for all } i \in \mathcal{C} \tag{16c}$$

$$R_i \leq y_i \leq H - R_i \qquad \text{for all } i \in \mathcal{C} \tag{16d}$$

$$z_i \in \{0, 1\} \qquad \text{for all } i \in \mathcal{C} \tag{16e}$$

where (x_i, y_i) is the center of circle $i \in \mathcal{C}$ and z_i the decision variable whether circle i has been packed, i.e., $z_i = 1$.

After solving the continuous relaxation of (13), it might happen that z_P^* is fractional for a rectangular pattern $P \in \mathcal{RP}'$. In this case branching is required to ensure global optimality. A general branching strategy has been introduced by [32], which has been successfully used in a branch-and-price algorithm for the one-dimensional cutting stock problem [30]. This branching rule can be applied when solving (13), however, it increases the complexity of the pricing problems (15). Since (15) is very difficult to solve—the vast majority of pricing problems cannot be solved to optimality in the root node—employing branch-and-price is deemed impractical. As such, Section 6.3 presents a *price-and-verify* algorithm, which builds upon price-and-branch, as a practical method for solving the RCPP. With techniques discussed in Section 6.2 this results in an algorithm that is able to prove global optimality for many RCPP instances.

5.4 Strength of Dantzig-Wolfe reformulations

In the following, we show that the two presented formulations (11) and (13) provide the same LP bound.

Theorem 1. Let LP(DW) and LP(PDW) be the value of the LP relaxation of (11) and (13), respectively. Then LP(PDW) = LP(DW).

Proof. To show the equality of the LP bounds, we consider an "extended" Dantzig-Wolfe reformulation. As a generalization of the variables z_F from (11), where $F \in \mathcal{F}$ encodes a rectangle of recursively packable rings, and z_P from (13), where $P \in \mathcal{RP}$ encodes a rectangular pattern, we introduce variables for "mixed" rectangles that may contain both rings and circles. Let $z_{(F,P)}$ be the number of such "mixed" rectangles, and denote with

$$\mathcal{MP} = \{(F, P) \in \mathbb{Z}_+^T \times \mathbb{Z}_+^T : (F, P) \text{ is packable } \}$$

the set of mixed rectangles. Here, (F, P) is packable if F_t rings of type t and P_t circles of type t can be packed into one rectangle without overlap, collectively over all t. A ring may contain smaller rings and circles, but circles cover their full interior, see Figure 6 for an example.

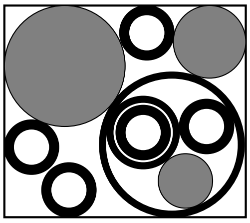

Figure 6: An example for a mixed rectangle packed with rings and circles.

Then define the LP

$$
\begin{aligned}
& \min \sum_{(F,P)\in\mathcal{MP}} z_{(F,P)} && && \text{(17a)}\\
\text{s.t.} & \sum_{(F,P)\in\mathcal{MP}} F_t \cdot z_{(F,P)} + \sum_{C=(t,P)\in\mathcal{CP}} z_C \geq D_t && \text{for all } t \in \mathcal{T} && \text{(17b)}\\
& \sum_{(F,P)\in\mathcal{MP}} P_t \cdot z_{(F,P)} + \sum_{C\in\mathcal{CP}} P_t \cdot z_C \geq \sum_{C=(t,P)\in\mathcal{CP}} z_C && \text{for all } t \in \mathcal{T} && \text{(17c)}\\
& z_C \geq 0 && \text{for all } C \in \mathcal{CP} && \text{(17d)}\\
& z_{(F,P)} \geq 0 && \text{for all } (F,P) \in \mathcal{MP} && \text{(17e)}
\end{aligned}
$$

and let Z^* be its optimal objective value. By construction, (17) contains the feasible solutions of (11) (setting $z_C = 0$ and $z_{(F,P)} = 0$ for all $P \neq 0$) and (13) (setting $z_{(F,P)} = 0$ for all $F \neq 0$). Hence, $Z^* \leq \mathrm{LP}(DW)$ and $Z^* \leq \mathrm{LP}(PDW)$. We prove the converse by optimality-preserving exchange arguments. First, given a solution with rectangles that contain rings, these rings can gradually be replaced by circles and circular patterns. Second, given a solution with rectangles that contain circles, we are guaranteed to find a circular pattern that can take its place. Formally, the proof reads as follows.

1. Suppose $\mathrm{LP}(PDW) > Z^*$ and let z^* be an optimal solution to (17) such that the total number of rings used in the rectangles of the support,

$$\mu(z) = \sum_{(F,P)\in\mathcal{MP}} \sum_{t=1}^{T} F_t \cdot z_{(F,P)}$$

is minimal. By assumption, $\mu(z^*) > 0$, and we can choose $(\tilde{F}, \tilde{P})$ and $\tilde{t}$ such that $z^*_{(\tilde{F},\tilde{P})} > 0$ and $\tilde{F}_{\tilde{t}} \geq 1$. Choose ring type $\tilde{t}$ to have smallest external radius $R_{\tilde{t}}$. Then this ring can only contain circles, and with these circles it forms a circular pattern $\tilde{C} = (\tilde{t}, P') \in \mathcal{CP}$. Replacing the ring of type $\tilde{t}$ by a circle with same external radius gives the packable tuple $(\hat{F}, \hat{P}) := (\tilde{F} - e_{\tilde{t}}, \tilde{P} + e_{\tilde{t}})$. Then define a new solution $\hat{z}$ via

$$
\hat{z}_{(F,P)} := \begin{cases} 0 & \text{if } (F,P) = (\tilde{F},\tilde{P}),\\ z^*_{(\hat{F},\hat{P})} + z^*_{(\tilde{F},\tilde{P})} & \text{if } (F,P) = (\hat{F},\hat{P}),\\ z^*_{(F,P)} & \text{otherwise,} \end{cases}
$$

for all $(F,P) \in \mathcal{MP}$, and

$$
\hat{z}_C := \begin{cases} z^*_{\tilde{C}} + z^*_{(\tilde{F},\tilde{P})} & \text{if } C = \tilde{C},\\ z^*_C & \text{otherwise,} \end{cases}
$$

for all $C \in \mathcal{CP}$. By construction, $\hat{z}$ is feasible for (17) and has identical objective function value. However, $\mu(\hat{z}) = \mu(z^*) - z^*_{(\tilde{F},\tilde{P})} < \mu(z^*)$, contradicting the minimality assumption.

2. Suppose $\mathrm{LP}(DW) > Z^*$ and let z^* be an optimal solution to (11) such that $\nu(z) = \sum_{C\in\mathcal{CP}} z_C$ is minimal. By assumption, $\nu(z^*) > 0$; otherwise, z^* could be transformed into an optimal solution of $\mathrm{LP}(DW)$ by removing all circles from each rectangle. Hence, we can choose a circular pattern $\tilde{C} = (\tilde{t}, P')$ with $z^*_{\tilde{C}} > 0$ and largest external radius $R_{\tilde{t}}$.

Now consider Constraint (17c) for $t = \tilde{t}$. Because $\sum_{C\in\mathcal{CP}} P_{\tilde{t}} \cdot z_C$ must be zero, there exists an $(\tilde{F}, \tilde{P})$ such that $z^*_{(\tilde{F},\tilde{P})} > 0$ and $\tilde{P}_{\tilde{t}} \geq 1$. Replacing one ring of type $\tilde{t}$ by the circular pattern $\tilde{C} = (\tilde{t}, P')$ gives the packable tuple $(\hat{F}, \hat{P}) := (\tilde{F} + e_{\tilde{t}}, \tilde{P} - e_{\tilde{t}} + P')$. Finally, we define a new solution $\hat{z}$ via

$$\hat{z}_{(F,P)} := \begin{cases} 0 & \text{if } (F,P) = (\tilde{F},\tilde{P}), \\ z^*_{(\hat{F},\hat{P})} + z^*_{(\tilde{F},\tilde{P})} & \text{if } (F,P) = (\hat{F},\hat{P}), \\ z^*_{(F,P)} & \text{otherwise,} \end{cases}$$

for all $(F, P) \in \mathcal{MP}$, and

$$\hat{z}_C := \begin{cases} 0 & \text{if } C = \tilde{C}, \\ z^*_C & \text{otherwise,} \end{cases}$$

for all $C \in \mathcal{CP}$. By construction, $\hat{z}$ is feasible for (17) and has identical objective function value. However, $\nu(\hat{z}) = \nu(z^*) - z^*_{\tilde{C}} < \nu(z^*)$, contradicting the minimality assumption. □ □

While (13) does not improve the strength of the LP relaxation, compared to (11), the advantage of (13) is that it breaks the symmetry of the combinatorial part of the RCPP, i.e., packing rings into rings, on each recursion level. Applying the pattern-based Dantzig-Wolfe reformulation makes deciding how to pack rings a counting problem in the master problem. Compared to (12), the resulting sub-problems do not contain the recursive structure of RCPP any more. This balances the complexity between the master problem and sub-problems, which is crucial for the performance of a column generation algorithm.

6 Column Generation Method for Solving the RCPP

This section presents a column generation based method for solving formulation (13). The key ingredients of our method are the enumeration of valid circular patterns, the generation of rectangular patterns, and a dynamic verification for circular pattern candidates during the column generation algorithm. Each of these fundamental components of the column generation based method are necessary for addressing the complexity of solving the pricing problem (15) and the difficulty in verifying whether a circular pattern candidate is packable. Since it may not be possible to compute all possible circular and rectangular patterns, the dynamic verification provides the capability to only check patterns that may be part of an optimal solution. Additionally, a key feature of our algorithm is that it is always capable of computing valid primal and dual bounds even though not all rectangular and circular patterns have been found.

6.1 Enumeration of Circular Patterns

Formulation (13) contains one variable for each circular pattern in $\mathcal{CP}$. This set is, in general, of exponential size. We present a column enumeration algorithm to compute all relevant circular patterns for (13). The main step of the algorithm is to verify whether a given tuple $(t, P) \in \mathcal{T} \times \mathbb{Z}_+^T$ is in the set $\mathcal{CP}$ or not. A tuple can be checked by solving the following nonlinear nonconvex verification problem:

$$\left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\|_2 \geq R_i + R_j \text{ for all } i, j \in C : i < j \tag{18a}$$

$$\left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right\|_2 \leq r_t - R_i \text{ for all } i \in C \tag{18b}$$

$$x_i, y_i \in \mathbb{R} \text{ for all } i \in C \tag{18c}$$

Here $C := \{1, \ldots, \sum_i P_i\}$ is the index set of individual circles, and R_i the corresponding external radius of a circle $i \in C$. Model (18) checks whether all circles can be placed in a non-overlapping way into a ring of type $t \in \mathcal{T}$. Constraint (18a) ensures that no two circles overlap, and (18b) guarantees that all circles are placed inside a ring of type t.

The non-overlapping conditions (18a) are nonconvex and make (18) computationally difficult to solve. Due to the positioning of circles, (18) contains a lot of symmetry. The rotation of every solution by 180° leads to another equivalent solution. Typically, this kind of symmetry is difficult to address within a global NLP solver. Branching on some continuous variables is likely to have no impact on the dual bound. One way to overcome this problem is to break some symmetry of the problem by ordering circles of the same type in the x-coordinate non-decreasingly. We achieve this by adding

$$x_i \leq x_j \text{ for all } i < j : R_i = R_j \tag{19}$$

to (18). Additionally, we add an auxiliary objective function $\min_{i \in C} x_i$ to (18). From our computational experiments we have seen that adding (19) to (18) makes it easier for the NLP solver to prove infeasibility.

As already seen in Section 5.1, there is a dominance relation between circular patterns, meaning that in any optimal solution of RCPP, dominated patterns can be replaced by non-dominated ones. Definition 1 formalizes this notion of a dominance relation between circular patterns.

Definition 1. A circular pattern $(t, P) \in \mathcal{CP}$ *dominates* $(t, P') \in \mathcal{CP}$ if and only if $P' <_{lex} P$, where $<_{lex}$ denotes the standard lexicographical order of vectors.

Let

$$\mathcal{CP}^* := \{(t, P) \in \mathcal{CP} \mid \nexists (t, P') \in \mathcal{CP} : (t, P') \text{ dominates } (t, P)\}$$

be the set of *non-dominated circular patterns.* The set $\mathcal{CP}^*$ might be much smaller than $\mathcal{CP}$, but is, in general, still of exponential size. Using $\mathcal{CP}^*$ in (13), instead of the larger set $\mathcal{CP}$, results in fewer variables.

Finally, we present the procedure *EnumeratePatterns* in order to compute $\mathcal{CP}^*$. Algorithm 1 considers all possible $(t, P) \in \mathcal{T} \times \mathbb{Z}_+^T$ and checks whether (t, P) is a circular pattern by solving (18). The algorithm exploits the dominance relation between circular patterns to reduce the number of NLP solves and filter dominated patterns. In the following, we discuss the different steps of Algorithm 1 in more detail. For simplicity, we define the set $[x] := \{0, \ldots, x\}$ for an integer number $x \in \mathbb{Z}_+$, and call a candidate pattern $(t, P) \in \mathcal{T} \times \mathbb{Z}_+^T$ *infeasible* if $(t, P) \notin \mathcal{CP}$ and feasible otherwise.

The algorithm maintains three sets $\mathcal{CP}_{feas}$, $\mathcal{CP}_{infeas}$, $\mathcal{CP}_{unknown}$, initialized to the empty set. In Line 3, we iterate through all possible pattern candidates (t, P) for a fixed $t \in \mathcal{T}$. In Line 5, we check whether P dominates a circular pattern already verified as infeasible and whether P is dominated by a circular pattern already verified as feasible. In both cases, P can

Algorithm 1: EnumeratePatterns

in : internal and external radii r and R, demands D
out: $\mathcal{CP}_{feas} \subseteq \mathcal{CP}$, unverified candidates $\mathcal{CP}_{unknown}$
1 $\mathcal{CP}_{feas} := \emptyset$, $\mathcal{CP}_{infeas} := \emptyset$, $\mathcal{CP}_{unknown} := \emptyset$
2 **for** $t \in \mathcal{T}$ **do**
3   **for** $P \in [D_1] \times \ldots \times [D_T]$ **do**
4   **if** $\exists (P', t) \in \mathcal{CP}_{infeas} : P' <_{lex} P$ *or* $\exists (P', t) \in \mathcal{CP}_{feas} : P' >_{lex} P$ **then**
5   **continue**
6         $status :=$ solve verification NLP (18)
7   **if** $status =$ *"feasible"* **then**
8             $\mathcal{CP}_{feas} := \mathcal{CP}_{feas} \cup (P, t)$
9   **if** $status =$ *"infeasible"* **then**
10             $\mathcal{CP}_{infeas} := \mathcal{CP}_{infeas} \cup (P, t)$
11   **if** $status =$ *"time limit" or "memory limit"* **then**
12             $\mathcal{CP}_{unknown} := \mathcal{CP}_{unknown} \cup (P, t)$
13 filter all dominated patterns from $\mathcal{CP}_{feas}$ and $\mathcal{CP}_{unknown}$
14 **return** $(\mathcal{CP}_{feas}, \mathcal{CP}_{unknown})$

be skipped. Otherwise, in Line 6, we solve a nonconvex verification NLP (18), which is the bottleneck of Algorithm 1. Roughly speaking, this NLP is easy to solve for the case where P, selected in Line 3, is component-wise too small or too large. In the first case, finding a feasible solution is easy due to a small number of circles. In the other case, we can conclude that the circles of the candidate P cannot be packed due to limited volume of the surrounding ring. The computationally expensive verification NLPs lie between these two extreme cases. Because of the two reversed dominance checks, it is in general unclear in which order the P in Line 3 should be enumerated. On the one hand, it can be beneficial to start with candidates that contain many circles. In case of a feasible candidate, many other candidates can be discarded because of the dominance relation between circular patterns. On the other hand, an infeasible candidate that dominates another infeasible candidate can incur a redundant, difficult NLP (18) solve.

Algorithm 1 returns two sets of circular patterns. The first set contains all feasible circular patterns that could be successfully verified and is denoted by $\mathcal{CP}_{feas}$. The second set, $\mathcal{CP}_{unknown}$, contains all candidates that could not be verified because of working limits, e.g., a time or memory limit on the solve of (18). At the end, each non-dominated circular pattern is either in the set $\mathcal{CP}_{feas}$ or $\mathcal{CP}_{unknown}$, which means that

$$\mathcal{CP}^* \subseteq \mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}$$

holds.

Ideally, we have verified all candidates, i.e., $\mathcal{CP}_{unknown} = \emptyset$. However, since there are exponentially many candidates to check and each candidate requires to solve (18), it might not be possible to compute $\mathcal{CP}^*$ in a reasonable amount of time.

Nevertheless, even if the set $\mathcal{CP}_{unknown}$ is non-empty, we are able to compute valid dual and primal bounds for RCPP. A valid primal bound is given when solving (13) after restricting the set of circular patterns to $\mathcal{CP}_{feas}$. This is becasuse this restriction ensures that we only use packable patterns. Using $\mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}$ in (13) yields a valid relaxation of RCPP because we use at least all patterns in $\mathcal{CP}^*$ and maybe some circular patterns that are not packable.

6.2 Computation of Valid Dual Bounds

The Pricing Problem (15) is a classical CPP—an $\mathcal{NP}$-hard nonlinear optimization problem, which is in practice very hard to solve. State-of-the-art MINLP solvers might fail to prove global optimality for this type of problems. Nevertheless, the following theorem shows how to use a dual bound for (15) to compute a valid dual bound for RCPP.

Theorem 2 (Farley [10], Vance et al. [31]). Let ν_{RMP} be the optimum of the restricted master problem of $PDW(\mathcal{RP}')$, $\nu_{Pricing}$ be the optimal value for the Pricing Problem (15), and OPT be the optimal solution value of RCPP. Then the inequality

$$\left\lceil \frac{\nu_{RMP}}{1 - z_{Pricing}} \right\rceil \leq OPT$$

holds for all valid dual bounds $z_{Pricing} \leq \nu_{Pricing}$.

In our computational experiments we have seen that this bound can indeed be used to obtain good quality bounds in cases where the circle packing pricing problem (16) could not be solved to optimality. Note that the dual bounds of Theorem 2 depend on the quality of the dual bounds of pricing problems. Any improvement on the dual bound for the CPP automatically translates to better dual bounds for the RCPP.

6.3 Price-and-Verify Algorithm

The price-and-verify algorithm is a column generation based algorithm that incorporates ideas from Sections 5, 6.1, and 6.2. This is summarized in Algorithm 2 and consists of three main steps:

1. An initial enumeration of circular patterns.
2. Generation of rectangular patterns with negative reduced cost.
3. Verification of circular pattern candidates during the pricing loop.

First, in Line 1, we use Algorithm 1 to compute the set of non-dominated circular patterns $\mathcal{CP}^*$. Its computational cost depends on the number of different ring types T, the external radii R, and the internal radii r. Algorithm 1 returns two sets $\mathcal{CP}_{feas}$ and $\mathcal{CP}_{unknown}$ with the properties

$$\begin{aligned} \mathcal{CP}_{feas} &\subseteq \mathcal{CP} \text{ and} \\ \mathcal{CP}^* &\subseteq \mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}. \end{aligned}$$

A feature of Algorithm 2 is the ability to compute valid dual and primal bounds simultaneously. This is achieved by computing the primal and dual bounds while dynamically verifying circular pattern candidates in $\mathcal{CP}_{unknown}$ during the solving process. The simultaneous computation of bounds is an improvement over the methods previously discussed. In the previous section, the proposed methods separately compute valid primal and dual bound for RCPP by using $\mathcal{CP}_{feas}$ and $\mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}$ respectively.

Algorithm 2 uses $\mathcal{CP}_{feas} \cup \mathcal{CP}_{unknown}$ as the initial set of circular patterns in Formulation (13). This ensures that at least all packable circular patterns are considered, which is necessary for proving a valid dual bound for RCPP. In general, many pattern candidates in $\mathcal{CP}_{unknown}$ are not packable and need to be discarded to prove global optimality. Algorithm 2 dynamically verifies $C \in \mathcal{CP}_{unknown}$ and fixes the corresponding variable z_C to zero if C is not packable. By discarding non-packable pattern candidates, the verification step does not only improve the quality of the dual bound but also ensures that any integer feasible solution $\bar{z}$ is feasible for the RCPP, i.e., $z_C = 0$ for all $C \in \mathcal{CP}_{unknown}$.

The key idea of the dynamic verification is to only consider candidate patterns in $\mathcal{CP}_{unknown}$ that have a nonzero LP solution value. To be more precise, let z^* be the optimal LP solution of the restricted master problem after no more rectangular patterns with negative reduced cost could be found in the pricing loop, see Line 3. Algorithm 2 solves the LP relaxation of the master problem (13) to optimality if $z^*_C = 0$ holds for all $C \in \mathcal{CP}_{unknown}$. Otherwise, there exists at least one $C \in \mathcal{CP}_{unknown}$ with $z^*_C > 0$. In order to verify C, we solve (18) in Line 6 with larger working limits than we have used in the initial enumeration step. There are three possible outcomes. The candidate pattern C

- **is packable:** We remove C from $\mathcal{CP}_{unknown}$, add it to $\mathcal{CP}_{feas}$, and continue with the next pattern candidate $C \in \mathcal{CP}_{unknown}$ that has a nonzero solution value in z^*.
- **is not packable:** We remove C from $\mathcal{CP}_{unknown}$ and fix z_C to zero, which cuts off the LP solution z^*. Resolving the LP leads to a different dual solution that might allow us to find new rectangular patterns with negative reduced cost. In this case, Algorithm 2 goes back to Line 3 and continues with pricing.
- **could not be verified:** Due to working limits, it may not be possible to verify C. In this case, we label that C has been tested, i.e., $\Psi(C) = 1$, and continue with the next candidate $C' \in \mathcal{CP}_{unknown}$ that has not been labeled yet, i.e., $\Psi(C') = 0$, and $z^*_{C'} > 0$. If C' can be verified to be not packable, we might get a different LP solution with $z^*_C = 0$. This would allow us to solve the LP relaxation of the master problem to optimality even though we could not verify C.

  In Line 16 we check whether there is still a candidate $C' \in \mathcal{CP}_{unknown}$ with $z^*_{C'} > 0$ left. The candidates where $z^*_{C'} > 0$ have already been tested, so $\Psi(C') = 1$, but were unable to be verified within the working limits. We fix all variables $z_{C'}$ to zero for all $C' \in \mathcal{CP}_{unknown}$. Since unverified patterns have been fixed to zero, it is no longer possible to compute a valid dual bound for RCPP. The remaining solution process can be seen as solving a restricted version of RCPP whose solution $\bar{z}$ is feasible for the original problem. However, it might be the case that $\bar{z}$ is still optimal, even though unverified candidates from $\mathcal{CP}_{unknown}$ are fixed to zero. It is possible to verify the circular patterns a posteriori to determine whether an optimal solution has been found.

The advantage of dynamically verifying circular pattern candidates during the pricing loop is that small working limits can be used in Algorithm 1 in order to identify packable patterns quickly and then focus, with larger working limits, on the patterns that are used in the LP solution of the restricted master problem.

After applying Algorithm 2 we have solved the LP relaxation of (13) in the root node of the branch-and-bound tree. If there exist any integer variables with a fractional solution value, then branching must be performed. Due to the complexity of the pricing problems, we observed in our experiments that solving (15) to global optimality is only possible for simple problems that can be solved in the root node—without requiring branching. The proposed branching strategy does not greatly reduce the complexity of the pricing problems at each node of the tree. As a result, performing pricing in each node is very time consuming. Thus, to improve the computational performance, pricing is only performed in the root node. Afterwards the RCPP is solved for the set of rectangular patterns that have been found so far. The optimal solution of this restricted problem is feasible for the original RCPP. Our results show that applying this strategy allows us to find good quality solutions for difficult problems.

Algorithm 2: Price-and-verify

in : internal and external radii r and R, demands D
out: LP solution z^* of the master problem or $\emptyset$

$(\mathcal{CP}_{feas}, \mathcal{CP}_{unknown}) :=$ EnumeratePatterns(r, R, D)
$\Psi_C := 0$ for all $C \in \mathcal{CP}_{unknown}$
while $\exists R \in \mathcal{RP} : red_R < 0$ **do**
  $\mathcal{RP} := \mathcal{RP} \cup \{R\}$ `// pricing loop`
$z^* :=$ solve LP(RMP)
while $\exists C \in \mathcal{CP}_{unknown} : z^*_C > 0 \wedge \Psi_C = 0$ **do**
  $status :=$ solve verification NLP (18) `// verification step`
  $\Psi_C := 1$
  if $status =$ *"feasible"* **then**
    $\mathcal{CP}_{unknown} := \mathcal{CP}_{unknown} \setminus \{C\}$
    $\mathcal{CP}_{feas} := \mathcal{CP}_{feas} \cup \{C\}$
  if $status =$ *"infeasible"* **then**
    $\mathcal{CP}_{unknown} := \mathcal{CP}_{unknown} \setminus \{C\}$
    fix $z_C := 0$ `// fixing cuts of` z^*
    go to 3 `// enter pricing loop again`
if $\exists C \in \mathcal{CP}_{unknown} : z^*_C > 0$ **then**
  fix $z_C := 0$ for all $C \in \mathcal{CP}_{unknown}$
  return $\emptyset$ `// LP solution is not valid`
return z^*

7 Computational Experiments

In this section, we investigate the performance and the quality of the dual and primal bounds obtained by our method and analyze how they relate to specific properties of an instance. The algorithm presented in Section 5.3 is implemented in the MINLP solver SCIP [29]. We refer to [2, 34] for an overview of the general solving algorithm and MINLP features of SCIP.

7.1 Implementation

We extended SCIP with the addition of two plug-ins: one pricing plug-in for solving the LP relaxation of Formulation (13) and one constraint handler plug-in to apply the dynamic verification of circular patterns during Algorithm 2. Algorithm 1 is executed immediately before the solving process for the RCPP commences. To accelerate the verification of circular pattern candidates, we use a simple greedy heuristic to check whether a given candidate (t, P) is a circular pattern, i.e., if $(t, P) \in \mathcal{CP}$. The heuristic iteratively packs circles to the left-most, and then lowest possible position in a ring of type $t \in \mathcal{T}$. If the heuristic fails to verify a candidate, we solve (18) until a feasible solution has been found, or it has been proven to be infeasible. The same heuristic is used for finding a rectangular pattern with negative reduced cost during Algorithm 2. If the heuristic fails to find such a pattern, we directly solve (15) to global optimality. The implementation is publically available in source code as part of the SCIP Optimization Suite and can be downloaded at https://scip.zib.de/.

7.2 Experimental Setup

We conducted three main experiments. In the first experiment we characterize instances for which Algorithm 1 finds all elements of the set of non-dominated circular patterns $\mathcal{CP}^* \subseteq \mathcal{CP}_{feas}$. The second experiment answers the question whether our proposed method is able to solve instances to global optimality and characterizes these instances by their structural properties. Because our method can also be used as a primal heuristic, in the last experiment we compare it with the GRASP heuristic of Pedroso et al. [25].

In the enumeration experiment we apply Algorithm 1 on each instance and check whether $\mathcal{CP}^*$ could be computed in two hours. For very difficult problems it might happen that we spend the whole time limit in solving a single NLP.

For our second experiment, we use our method to compute valid dual and primal bounds for RCPP with a total time limit of two hours. In contrast to the first experiment, we enforce a time limit of $10s$ for each NLP (18) in Algorithm 1. After a time budget of $1200s$, we stop solving (18) and only use the greedy heuristic to verify circular pattern candidates. A pattern is added to $\mathcal{CP}_{feas}$ if it can be verified. Otherwise, we add a candidate to $\mathcal{CP}_{unknown}$ and process with the next candidate pattern. During the pricing loop we then use a larger time limit of $120s$ to verify a pattern in $\mathcal{CP}_{unknown}$ that has a nonzero value in the LP relaxation solution. Again, we stop solving NLPs after $2400s$ were spent on verifying pattern candidates.

During Algorithm 2, we use a time limit of $300s$ to solve (15). If we fail to solve a pricing problem to optimality and no improving column could be found, we stop solving any further pricing problems, obtain a valid dual bound by applying Theorem 2, and continue to solve the restricted master problem for the current set of rectangular patterns. In our experiments, this only occurs at the root node.

Finally, in our third experiment we compare the obtained primal bounds of our method with those obtained from the GRASP heuristic. Both algorithms run with a time limit of three hours on each instance. For this experiment the specifications from the second experiment are used for out method. We use the Python implementation of GRASP from [25]. Note that SCIP is written in the programming language C, in which an implementation of GRASP would be much faster. However, in our primal bound experiment we only compare the quality of obtained solutions and it can be observed that GRASP finds its best solutions in the first few minutes.

Test Sets. We consider two different test sets for our experiments. The first one contains 9 real-world instances from the tube industry, which were used in [25], and is in the following called REAL test set. Because this test set is too limited for a detailed computational study, we created a second test set containing 800 instances, the RAND test set. The purpose of this set is to show how the number of different ring types T, the maximum ratio of external radii $\max_t r_t / \min_t R_t$, and the ratio between rectangle size and the maximum volume of a ring $\max\{W, H\}/\max_t \pi(R_t)^2$ influence the performance of our method.

The name of each instance reads `i<T>_<α>_<β>_<γ>.rpa` where

- $T \in \{3, 4, 5, 10\}$ is the number of ring types,
- $\frac{\max_t r_t}{\min_t R_t} =: \alpha \in \{2.0, 2.3, \ldots, 4.7\}$ is the maximum external radii ratio,
- $\frac{\max\{W,H\}}{\max_t R_t} =: \beta \in \{2.0, 2.3, \ldots, 4.7\}$ is the rectangle size to external radius ratio, and
- $\frac{WH}{\max_t \pi(R_t)^2} [0.8\gamma, 1.2\gamma]$ for $\gamma \in \{5, 10\}$ is the demand interval for each type t.

The demand of a type $t \in \mathcal{T}$ is randomly chosen from the corresponding demand interval. The size of this interval is anti-proportional to the external radius, i.e., types with large external radius appear less often. For all instances in RAND we fixed the width W and height H of the rectangles to 10. We created one instance for each 4-tuple, giving 800 instances in total. All instances of the RAND and REAL test set are publically available at https://github.com/mueldgog/RecursiveCirclePacking.

Hardware and Software. The experiments were performed on a cluster of 64bit Intel(R) Xeon(R) CPU E5-2660 v3 2.6 GHz with 12 MB cache and 48 GB main memory. In order to safeguard against a potential mutual slowdown of parallel processes, we ran only one job per node at a time. We used SCIP version 5.0 with CPLEX 12.7.1.0 as LP solver [18], CppAD 20140000.1 [5], and Ipopt 3.12.5 with MUMPS 4.10.0 [1] as NLP solver [35, 6].

7.3 Computational Results

In the following, we discuss the results for the three described experiments in detail. Instance-wise results of all experiments can be found in Table in the electronic supplement.

Enumeration Experiments. Figure 7 shows the computing time required to compute all non-dominated circular patterns $\mathcal{CP}^*$ for the instances of the RAND test set. Each line corresponds to the subset of instances with identical number of ring types T. Each point on a line is computed as the shifted geometric mean (with a shift of 1.0) over all instances that have the same value $\max_t r_t / \min_t R_t$. We expect that the number of circular patterns increases when increasing the ratio between the ring with largest inner and the ring with smallest outer radius.

The first observation is that the time to compute all circular patterns increases when T increases. For example, for $T = 10$ we need about $4-10$ times longer to enumerate all patterns than for $T = 5$. Also, all lines in Figure 7 approximately increase exponentially in $\max_t r_t / \min_t R_t$. A larger ratio implies that each verification NLPs (18) has a larger number of circles that fit into a ring of inner radius $\max_t r_t$. These difficult NLPs and the larger number of possible circular patterns provide an explanation for the exponential increase of each line in Figure 7.

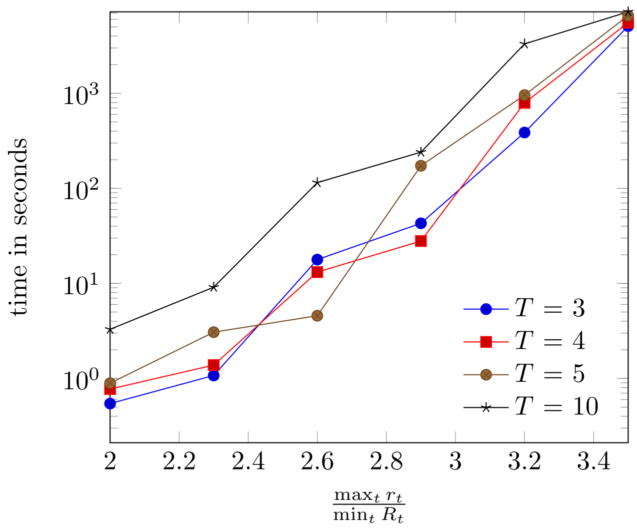

Figure 7: Plot showing the average time to enumerate all circular patterns for different number of ring types $T \in \{3, 4, 5, 10\}$ within a two hours time limit.

Table 1 shows in more detail for how many instances we could compute $\mathcal{CP}^*$ successfully and how much time it took on average. The rows partition the instances according to the number of ring types and the columns group these instances by their $\max_t r_t/\min_t R_t$ values. First, we see that for 392 of the 800 instances of the RAND test set we could successfully enumerate all circular patterns, in 19.8 seconds on average. We see that for less and less instances the enumeration step is successful as $\max_t r_t/\min_t R_t$ increases. For none of the RAND instances with $\max_t r_t/\min_t R_t$ strictly larger than 3.5 could $\mathcal{CP}^*$ be successfully enumerated. Many verification NLPs for these instances are too difficult to solve and often consume the whole time limit of two hours, even for the case of three different ring types. However, for the 480 instances with $\max_t r_t/\min_t R_t \leq 3.5$ Algorithm 1 managed to compute on 81.6% (392) of these instances all non-dominated circular patterns.

Table 1: Aggregated results for enumerating all non-dominated circular patterns. Each of the six columns reports the results for 80 instances of the RAND test set. The 320 instances with $\max_t r_t/\min_t R_t > 3.5$ are not shown because none of them could be successfully enumerated within the time limit.

n — number of instances for which $\mathcal{CP}^*$ could be computed
time — time in seconds

$\nu = \frac{\max_t r_t}{\min_t R_t}$	$\nu = 2.0$		$\nu = 2.3$		$\nu = 2.6$		$\nu = 2.9$		$\nu = 3.2$		$\nu = 3.5$	
	n	time	n	time	n	time	n	time	n	time	n	time
$T = 3$	20	0.5	20	1.0	20	8.1	20	35.2	17	227.5	12	4050.7
$T = 4$	20	0.7	20	1.3	20	8.2	20	24.2	15	374.5	9	4020.3
$T = 5$	20	0.9	20	2.6	19	4.3	19	117.9	16	571.4	4	4507.7
$T = 10$	20	3.0	20	8.0	16	28.9	17	117.3	8	1022.6	0	–
all	80	1.1	80	2.5	75	9.3	76	56.8	56	419.4	25	4109.4

Exact Price-and-Verify. In our second experiment, we analyze the primal and dual bounds that have been computed by our column generation method. Figure 8 shows the achieved optimality gaps, i.e., (primal − dual)/dual, for all instances of the RAND test set. We solve 35.9% of the instances to global optimality and get gaps between 0-25% for 5.6% of the instances. For about 46.3% of the instances we achieve gaps between 25-100%, and only for a single instance the gap is larger than 100%.

Table 2 and Table 3 contain aggregated results for the optimality gaps for the RAND test set. Table 2 shows that out of the 287 optimally solved instances, 80 instances have three, 72 have four, 70 have five, and 65 have ten different ring types. Most of the instances with a gap larger than 50% are from the subset of instances with ten ring types. Only 33 of the 98 instances with a gap larger than 50% have less than ten different ring types. Figure 9 visualizes the results of Table 2 in a bar diagram. As expected, for an increasing number of ring types worse optimality gaps are obtained.

Each row in Table 3 corresponds to the set of instances that have at least a certain value for $\max_t r_t/\min_t R_t$. For example, the bottom-left corner value signifies that 60 out of the 160 instances with $\max_t r_t/\min_t R_t \geq 4.4$ could be solved to optimality. As for Table 2, we see that the gaps increase with a larger $\max_t r_t/\min_t R_t$ ratio. This can be explained by our previous observation, namely, the difficulty of enumerating all circular patterns for these instances. When $\mathcal{CP}_{unknown} \neq \emptyset$, then it could contain packable and unpackable patterns. A valid dual bound is given by the solution to the LP relaxation without any unverified patterns fixed to zero. If

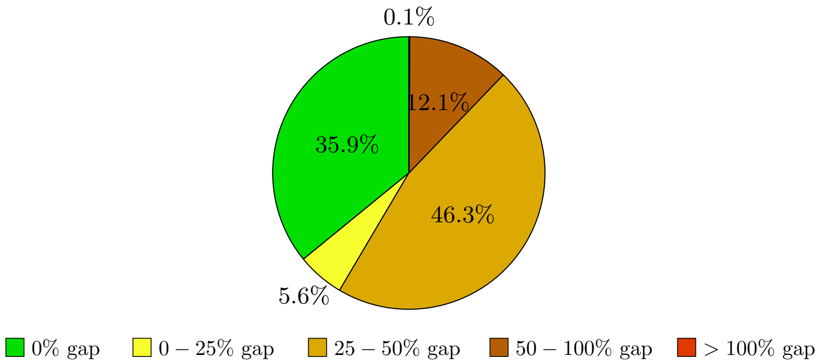

Figure 8: Optimality gaps for all RAND instances.

Table 2: Number of instances of the RAND test set according to final gap and number of ring types.

	total	0%	0 – 25%	25 – 50%	50 – 100%	> 100%
$T = 3$	200	80	14	99	7	0
$T = 4$	200	72	11	108	9	0
$T = 5$	200	70	9	104	17	0
$T = 10$	200	65	11	59	64	1
all	800	287	45	370	97	1

this solution contains patterns from $\mathcal{CP}_{unknown}$ that are unpackable, then the computed bound will be lower than the optimal LP bound if all circular patterns were verified. Alternatively, after the LP relaxation has been solved to optimality, all unverified patterns are fixed to zero. If $\mathcal{CP}_{unknown}$ contains packable patterns that are necessary to express an optimal integer solution, then the best primal bound will be greater than the optimal solution value to the RCPP. To summarize, our results show that the number of ring types T and the quotient $\max_t r_t / \min_t R_t$ are in many cases reliable indicators for both the quality of primal and dual bounds and the computational cost of our method. Instances that require branching are often too difficult to solve to global optimality.

Table 3: Number of instances of the RAND test set according to final gap and $\max_t r_t / \min_t R_t$.

	total	0%	0 – 25%	25 – 50%	50 – 100%	> 100%
$\max_t r_t / \min_t R_t \geq 2.0$	800	287	45	370	97	1
≥ 2.6	640	223	42	295	79	1
≥ 3.2	480	170	36	204	69	1
≥ 3.8	320	122	25	126	46	1
≥ 4.4	160	60	11	66	23	0

Finally, we briefly report on the size of the branch-and-bound trees for the primal and dual bound experiments. We consider a node to be explored once it has been selected by SCIP's node

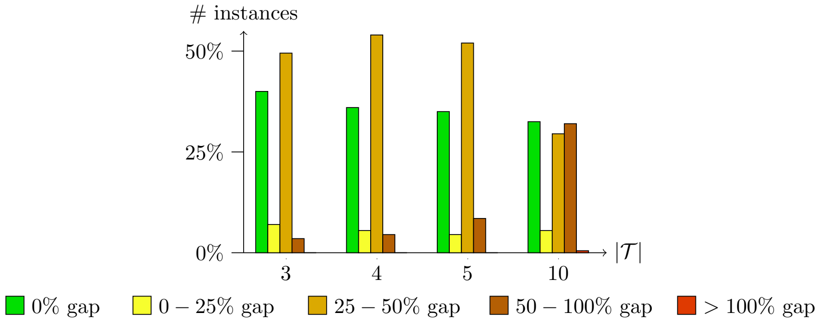

Figure 9: Optimality gaps for RAND instances split by number of different ring types.

selection algorithm. Our method explored more than one node for 367 of the 800 RAND instances and only 15 of these instances could be solved to global optimality. The maximum number of explored nodes is 207.

Primal Bound Experiments. In our final experiments, we consider our method as a pure primal heuristic for RCPP and compare it to the GRASP heuristic by Pedroso et al. [25], which is specifically designed to find dense packings quickly. Table 4 contains detailed results for the REAL, and Table 5 aggregated results for the RAND test set. For analytical purposes, Table 4 also shows the results for the dual bounds.

Table 4: Detailed results for the comparison with GRASP on the REAL test set.

Instance				price & verify		GRASP
Name	T	$\frac{\max_t r_t}{\min_t R_t}$	volume	dual	primal	primal
s03i1	3	2.2	1	1	1	1
s03i2	3	2.2	5	9	10	10
s03i3	3	2.2	47	82	98	95
s05i1	5	5.1	1	1	1	1
s05i2	5	5.1	4	8	10	10
s05i3	5	5.1	36	70	94	98
s16i1	16	6.3	1	1	1	1
s16i2	16	6.3	5	8	10	10
s16i3	16	6.3	45	73	96	96

For the REAL test set, we are able to solve the three instances `s**i1`, that require only one rectangle in the optimal solution. On all but two instances our method achieves the same primal bound as GRASP. On `s03i3` we need three more rectangles and on `s05i3` four rectangles less. Due to the small value $\max_t r_t / \min_t R_t = 2.2$ of the instances with three different ring types, the enumeration step takes only a fraction of a second. The difficulty of these instances lies only in the placement of rings into rectangles and not in the recursive structure of RCPP. Hence, it is not surprising that our method computes a worse solution than GRASP on `s03i3` because we only use a simple left-most greedy heuristic before solving (15).

Proving optimality for the instances of the REAL test set is difficult for our method. Only the three instances `s**i1` with one rectangle in the optimal packing, could be solved to gap zero. Because the rectangles are very large relative to the radii of the rings, the pricing problems contain many variables and constraints. As a result, the pricing problems are too difficult to solve to optimality. As shown above, the ratio $\max_t r_t / \min_t R_t$ of the instances with $T > 3$ is too large to enumerate and verify all circular patterns in reasonable time.

It is worth to notice that the dual bounds that can be proved by our method are much better than simple volume-based bounds. Except for the three instances that require only one rectangle, our dual bounds are at least 1.6 times larger than the volume-based bounds.

Even though our method is not particularly designed for the characteristics of the instances in the REAL test set, it still performs well as a primal heuristic. On the instances with five ring types it could enumerate many non-trivial circular patterns and uses them to find a better primal solution than GRASP.

For the RAND test set, Table 5 shows the number of instances on which our method performed better or worse than GRASP with respect to the achieved primal bound. The second column reports the primal bound average relative to the value of GRASP on all instances of RAND. For this we compute the shifted geometric mean of all ratios between the obtained primal bounds of our method and GRASP. The third column shows the total number of instances on which our method found a better primal solution. The fourth column shows the improvement relative to GRASP. The remaining three columns show statistics for the instances on which our method performed worse.

We observe that our method outperforms GRASP on many instances. On average, the solutions are 2.7% better than the ones from GRASP. In total, we find a better primal bound on 356 of the instances, and a worse bound on only 33 instances. Our method could find for all instances a primal feasible solution before hitting the time or memory limit. The average deterioration on the 33 instances is $5.1 - 7.2\%$ and the average improvement on the 356 instances is $5.6 - 6.9\%$. Our method finds more often a better solution than GRASP for instances that have a larger number of ring types. For ten ring types we find 123 better solutions, which is about twice as large as for three ring types.

There are two reasons why our method performs better than GRASP on many instances. The first is that GRASP considers and positions each ring individually. A large demand vector D results in a large number of rings, which makes it difficult for GRASP to find good primal solutions. In contrast, the number of rings and circles that need to be considered in our pricing problems and in the enumeration is typically bounded by a number derived from some volume arguments instead of the entries of the demand vector D. Thus, scaling up D typically does not have a large impact when applying our column generation algorithm as a primal heuristic. The second reason for the better performance is that for a given set of circular and rectangular patterns, the master problem takes the best decisions of how to pack rings recursively into each other. For instances where this combinatorial part of the problem is difficult, we expect that our method performs better than GRASP. Indeed, on instances with a large number of ring types our method frequently finds better solutions than GRASP.

In summary, the experiments on synthetic and real-world instances shows that our method solves small and medium-sized instances to global optimality. This was the case on 287 of the randomly generated instances and for three of nine real-world instances. Due to the costly verification of circular patterns and difficult sub-problems, our method is not able to solve larger instances to global optimality, but still achieves good primal and dual bounds. Compared to the state-of-the-art heuristic for RCPP, our method finds on 356 of the instances better, and only on 33% of the instances worse primal solutions.

Table 5: Aggregated results for primal bound comparisons against GRASP on the RAND test set.

all — all instances
better — instances with a better primal bound than GRASP
worse — instances with a worse primal bound than GRASP
— number of instances
% — average primal bound relative to average primal bound of GRASP

	all	better		worse	
	%	#	%	#	%
$T = 3$	98.3	61	94.3	7	106.3
$T = 4$	97.3	78	93.1	9	105.5
$T = 5$	97.4	94	94.4	9	107.2
$T = 10$	96.1	123	93.6	8	105.1
all	97.3	356	93.8	33	106.0

8 Conclusion

In this article, we have presented the first exact algorithm for solving practically relevant instances for the extremely challenging recursive circle packing problem. Our method is based on a Dantzig-Wolfe decomposition of a nonconvex MINLP model. The key idea of solving this decomposition via column generation is to break symmetry of the problem by using circular and rectangular patterns. These patterns are used to model all possible combinations of packing rings into other rings. As a result, we were able to reduce the complexity of the sub-problems significantly and shift the recursive part of RCPP to a linear master problem.

In some sense, this reformulation can be interpreted as a reduction technique from RCPP to CPP. All occurring sub-problems in the enumeration of circular patterns and the pricing problems are classical CPPs and they constitute the major computational bottleneck. Every primal or dual improvement for this problem class would directly translate to better primal and dual bounds for RCPP. Still, in order to prove optimality it is usually necessary to solve the $\mathcal{NP}$-hard CPP to optimality, which can fail for harder instances. However, even for this case, the application of Theorem 2 and the pessimistic and optimistic enumeration of circular patterns guarantees valid primal and dual bounds for RCPP. The combination of column generation with column enumeration could be of more general interest when using decomposition techniques for MINLPs that lead to hard nonconvex sub-problems.

An interesting extension of the presented method is its application to problem with different container shapes and sizes. Since the master problem of the decomposed model does not depend on the shape of the containers or packed objects, it is possible to apply this model to any container shape. This would only require a modification to the column generation subproblems to produce packable patterns that can be added to the master problem. Thus, the presented methods cover a wide range of packing problems.

Finally, our current proof-of-concept implementation could certainly be improved further. To mention only one point, we currently use a rather simple greedy heuristic in order to find quickly a feasible solution for the verification NLP (18) and the Pricing Problem (15). By using more sophisticated heuristics for the well-studied CPP, we might be able to compute better optimality gaps for instances where the positioning of circles into rectangles or rings is difficult. Surprisingly, even with a simple greedy heuristic our method works well as a primal heuristic and finds for many instances better solutions than GRASP [25].

Acknowledgments

This work has been supported by the Research Campus MODAL *Mathematical Optimization and Data Analysis Laboratories* funded by the Federal Ministry of Education and Research (BMBF Grant 05M14ZAM). All responsibility for the content of this publication is assumed by the authors.

References

[1] MUMPS, Multifrontal Massively Parallel sparse direct Solver. http://mumps.enseeiht.fr

[2] Achterberg, T.: Constraint integer programming. Ph.D. thesis, Technische Universität Berlin (2007). URN:nbn:de:kobv:83-opus-16117

[3] Bergner, M., Caprara, A., Ceselli, A., Furini, F., Lübbecke, M.E., Malaguti, E., Traversi, E.: Automatic Dantzig–Wolfe reformulation of mixed integer programs. Mathematical Programming **149**(1-2), 391–424 (2015)

[4] Castillo, I., Kampas, F.J., Pintér, J.D.: Solving circle packing problems by global optimization: Numerical results and industrial applications. European Journal of Operational Research **191**(3), 786–802 (2008). URL http://EconPapers.repec.org/RePEc:eee:ejores:v:191:y:2008:i:3:p:786-802

[5] COIN-OR: CppAD, a package for differentiation of CPP algorithms. http://www.coin-or.org/CppAD

[6] COIN-OR: Ipopt, Interior point optimizer. http://www.coin-or.org/Ipopt

[7] Costa, A., Hansen, P., Liberti, L.: On the impact of symmetry-breaking constraints on spatial branch-and-bound for circle packing in a square. Discrete Applied Mathematics **161**(1–2), 96 – 106 (2013). DOI http://dx.doi.org/10.1016/j.dam.2012.07.020. URL http://www.sciencedirect.com/science/article/pii/S0166218X12002855

[8] Desaulniers, G., Desrosiers, J., Solomon, M.M.: Column generation, vol. 5. Springer Science & Business Media (2006)

[9] DROR, M.: Polygon plate-cutting with a given order. IIE Transactions **31**(3), 271–274 (1999). DOI 10.1023/A:1007606007369. URL http://dx.doi.org/10.1023/A:1007606007369

[10] Farley, A.A.: A note on bounding a class of linear programming problems, including cutting stock problems. Oper. Res. **38**(5), 922–923 (1990). DOI 10.1287/opre.38.5.922. URL http://dx.doi.org/10.1287/opre.38.5.922

[11] Fortz, B., Labbé, M., Poss, M.: A branch-and-cut-and-price framework for convex minlp applied to a stochastic network design problem. In: Workshop on Mixed Integer Nonlinear Programming, p. 131 (2010)

[12] Gamrath, G.: Generic branch-cut-and-price. Master's thesis (2010)

[13] George B. Dantzig, P.W.: Decomposition principle for linear programs. Operations Research **8**(1), 101–111 (1960). URL http://www.jstor.org/stable/167547

[14] Gilmore, P., Gomory, R.: A linear programming approach to the cutting stock problem. Operations Research **9** (1961)

[15] Gilmore, P., Gomory, R.E.: Multistage cutting stock problems of two and more dimensions. Operations research **13**(1), 94–120 (1965)

[16] Hales, T.C., Adams, M., Bauer, G., Dang, D.T., Harrison, J., Hoang, T.L., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T.T., Nguyen, T.Q., Nipkow, T., Obua, S., Pleso, J., Rute, J., Solovyev, A., Ta, A.H.T., Tran, T.N., Trieu, D.T., Urban, J., Vu, K.K., Zumkeller, R.: A formal proof of the Kepler conjecture. arXiv **1501.02155** (2015)

[17] Hifi, M., M'Hallah, R.: A literature review on circle and sphere packing problems: Models and methodologies. Adv. Operations Research **2009**, 150,624:1–150,624:22 (2009). URL http://dblp.uni-trier.de/db/journals/advor/advor2009.html#HifiM09

[18] ILOG, Inc: ILOG CPLEX: High-performance software for mathematical programming and optimization. See http://www.ilog.com/products/cplex/

[19] Jans, R.: Solving lot-sizing problems on parallel identical machines using symmetry-breaking constraints. INFORMS Journal on Computing **21**(1), 123–136 (2009)

[20] Kallrath, J.: Cutting circles and polygons from area-minimizing rectangles. Journal of Global Optimization **43**(2), 299–328 (2009). DOI 10.1007/s10898-007-9274-6. URL http://dx.doi.org/10.1007/s10898-007-9274-6

[21] Kallrath, J.: Packing ellipsoids into volume-minimizing rectangular boxes. Journal of Global Optimization pp. 1–35 (2015). DOI 10.1007/s10898-015-0348-6. URL http://dx.doi.org/10.1007/s10898-015-0348-6

[22] Kilinç, M., Sahinidis, N.V.: Solving MINLPs with BARON. Talk at MINLP 2014, Carnegie Mellon University, Pittsburgh, PA, USA (2014). http://minlp.cheme.cmu.edu/2014/papers/kilinc.pdf

[23] Lenstra, J.K., Kan, A.H.G.R.: Complexity of packing, covering and partitioning problems. Econometric Institute (1979)

[24] Lübbecke, M.E., Desrosiers, J.: Selected topics in column generation. Operations Research **53**(6), 1007–1023 (2005)

[25] Pedroso, J.P., Cunha, S., Tavares, J.N.: Recursive circle packing problems. International Transactions in Operational Research **23**(1-2), 355–368 (2016). DOI 10.1111/itor.12107. URL http://dx.doi.org/10.1111/itor.12107

[26] Pisinger, D., Sigurd, M.: The two-dimensional bin packing problem with variable bin sizes and costs. Discret. Optim. **2**(2), 154–167 (2005). DOI 10.1016/j.disopt.2005.01.002. URL http://dx.doi.org/10.1016/j.disopt.2005.01.002

[27] Ralphs, T.K., Galati, M.V.: Decomposition in integer linear programming. Integer Programming: Theory and Practice **3** (2005)

[28] Sahinidis, N.V.: Baron: A general purpose global optimization software package. Journal of Global Optimization **8**(2), 201–205 (1996). DOI 10.1007/BF00138693. URL http://dx.doi.org/10.1007/BF00138693

[29] SCIP – Solving Constraint Integer Programs. http://scip.zib.de

[30] Vance, P.H.: Branch-and-price algorithms for the one-dimensional cutting stock problem. Computational Optimization and Applications **9**(3), 211–228 (1998). DOI 10.1023/A:1018346107246. URL http://dx.doi.org/10.1023/A:1018346107246

[31] Vance, P.H., Barnhart, C., Johnson, E.L., Nemhauser, G.L.: Solving binary cutting stock problems by column generation and branch-and-bound. Computational Optimization and Applications **3**(2), 111–130 (1994). DOI 10.1007/BF01300970. URL http://dx.doi.org/10.1007/BF01300970

[32] Vanderbeck, F., Wolsey, L.A.: An exact algorithm for ip column generation. Oper. Res. Lett. **19**(4), 151–159 (1996). DOI 10.1016/0167-6377(96)00033-8. URL http://dx.doi.org/10.1016/0167-6377(96)00033-8

[33] Vanderbeck, F., Wolsey, L.A.: Reformulation and Decomposition of Integer Programs, pp. 431–502. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). DOI 10.1007/978-3-540-68279-0_13. URL http://dx.doi.org/10.1007/978-3-540-68279-0_13

[34] Vigerske, S.: Decomposition in multistage stochastic programming and a constraint integer programming approach to mixed-integer nonlinear programming. Ph.D. thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II (2013). URN:nbn:de:kobv:11-100208240

[35] Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming **106**(1), 25–57 (2006). DOI:10.1007/s10107-004-0559-y

Appendix

Table 6: Detailed results for randomly generated instances.

vol — volume-based dual bound
dual — proven primal bound
primal — proven dual bound
GRASP — proven primal bound by GRASP heuristic
nfeas — total number of circular patterns found
ncands — total number of circular patterns candidates
nrp — total number of generated rectangular patterns
nodes — total number of branch-and-bound nodes
enum time — time to compute all circular patterns in enumeration experiment

instance	**vol**	**dual**	**primal**	**GRASP**	**nfeas**	**ncands**	**nrp**	**nodes**	**enum time**
i03_2.0_2.0_05	8	17	17	17	4	0	6	1	0.4
i03_2.0_2.0_10	22	34	38	38	5	0	7	3	1.4
i03_2.0_2.3_05	7	16	16	16	4	0	6	1	0.5
i03_2.0_2.3_10	23	33	45	49	4	0	9	3	0.9
i03_2.0_2.6_05	8	23	23	23	3	0	7	1	0.6
i03_2.0_2.6_10	14	41	41	41	4	0	9	1	0.3
i03_2.0_2.9_05	8	16	16	16	4	0	9	1	0.4
i03_2.0_2.9_10	14	18	27	27	4	0	8	3	0.7
i03_2.0_3.2_05	9	23	23	23	4	0	7	1	0.6
i03_2.0_3.2_10	20	26	38	38	4	0	11	3	1.0
i03_2.0_3.5_05	8	21	21	21	3	0	10	1	0.7
i03_2.0_3.5_10	16	23	32	33	4	0	10	3	0.5
i03_2.0_3.8_05	6	10	14	15	4	0	15	3	0.4
i03_2.0_3.8_10	16	23	31	33	4	0	16	3	0.3
i03_2.0_4.1_05	9	13	18	18	4	0	21	3	0.3
i03_2.0_4.1_10	12	18	24	24	4	0	25	3	0.4
i03_2.0_4.4_05	8	10	14	15	4	0	18	3	0.3
i03_2.0_4.4_10	14	20	25	27	4	0	18	3	0.3
i03_2.0_4.7_05	7	13	19	19	3	0	16	3	0.6
i03_2.0_4.7_10	14	21	29	30	4	0	24	3	0.4
i03_2.3_2.0_05	7	15	15	15	3	0	5	1	2.2
i03_2.3_2.0_10	16	36	36	36	4	0	6	1	0.5
i03_2.3_2.3_05	8	12	18	18	4	0	7	3	0.9
i03_2.3_2.3_10	14	32	32	32	4	0	5	1	1.2
i03_2.3_2.6_05	9	11	17	18	4	0	7	3	0.8
i03_2.3_2.6_10	16	30	35	38	4	0	10	4	1.0
i03_2.3_2.9_05	7	17	17	17	4	0	7	1	0.6
i03_2.3_2.9_10	17	66	66	66	3	0	5	1	2.1
i03_2.3_3.2_05	7	11	16	16	4	0	7	3	0.7
i03_2.3_3.2_10	14	38	38	38	4	0	5	1	0.9
i03_2.3_3.5_05	8	23	23	23	3	0	6	1	0.9
i03_2.3_3.5_10	14	21	28	29	4	0	14	3	0.9
i03_2.3_3.8_05	7	11	15	15	4	0	16	3	0.7
i03_2.3_3.8_10	23	25	34	35	5	0	18	20	2.6

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i03_2.3_4.1_05	7	10	13	13	4	0	12	3	1.1
i03_2.3_4.1_10	16	24	31	30	3	0	22	3	0.9
i03_2.3_4.4_05	8	11	15	15	4	0	21	3	0.5
i03_2.3_4.4_10	13	19	24	25	5	0	8	3	1.6
i03_2.3_4.7_05	9	13	19	20	3	0	10	3	0.8
i03_2.3_4.7_10	16	23	30	30	4	0	23	3	0.8
i03_2.6_2.0_05	9	14	20	19	4	0	5	3	1.5
i03_2.6_2.0_10	12	34	34	34	4	0	5	1	1.3
i03_2.6_2.3_05	7	11	16	14	4	0	5	3	1.6
i03_2.6_2.3_10	17	29	39	40	4	0	7	3	1.4
i03_2.6_2.6_05	8	25	25	25	3	0	3	1	2.2
i03_2.6_2.6_10	24	26	36	37	8	1	11	3	11.6
i03_2.6_2.9_05	8	30	30	30	3	0	3	1	3.2
i03_2.6_2.9_10	13	67	67	67	3	0	3	1	2.5
i03_2.6_3.2_05	6	19	19	19	4	0	5	1	1.6
i03_2.6_3.2_10	12	37	37	37	6	0	5	1	1599.4
i03_2.6_3.5_05	8	12	16	17	4	0	19	3	2.0
i03_2.6_3.5_10	24	26	34	35	8	1	15	3	5686.4
i03_2.6_3.8_05	6	8	11	11	4	0	11	3	1.7
i03_2.6_3.8_10	12	38	38	38	3	0	6	1	2.0
i03_2.6_4.1_05	7	9	13	12	5	0	8	3	4.6
i03_2.6_4.1_10	14	19	26	26	3	0	21	3	2.5
i03_2.6_4.4_05	7	12	15	15	3	0	17	3	1.9
i03_2.6_4.4_10	22	23	31	32	8	1	15	3	1499.0
i03_2.6_4.7_05	6	10	14	14	4	0	16	3	1.6
i03_2.6_4.7_10	13	18	24	24	5	0	18	3	4.7
i03_2.9_2.0_05	7	12	17	17	4	0	6	3	10.9
i03_2.9_2.0_10	13	29	29	29	3	0	3	1	18.2
i03_2.9_2.3_05	6	15	15	15	3	0	3	1	17.6
i03_2.9_2.3_10	14	40	40	40	3	0	3	1	15.1
i03_2.9_2.6_05	7	18	18	18	4	0	5	1	13.5
i03_2.9_2.6_10	13	41	41	41	3	0	3	1	21.5
i03_2.9_2.9_05	7	11	12	12	7	1	12	3	843.5
i03_2.9_2.9_10	13	66	66	66	3	0	3	1	811.0
i03_2.9_3.2_05	7	14	14	15	4	0	16	1	12.7
i03_2.9_3.2_10	14	38	38	38	5	1	5	1	15.3
i03_2.9_3.5_05	7	10	13	13	7	1	7	3	63.6
i03_2.9_3.5_10	20	20	29	30	10	3	17	1	3990.8
i03_2.9_3.8_05	7	10	13	13	4	0	22	3	9.7
i03_2.9_3.8_10	12	14	19	19	9	2	9	1	320.8
i03_2.9_4.1_05	7	10	13	13	4	0	17	3	10.1
i03_2.9_4.1_10	15	20	26	27	7	1	16	3	27.0
i03_2.9_4.4_05	8	9	12	12	4	0	17	3	12.3
i03_2.9_4.4_10	12	21	27	28	4	0	13	3	11.1
i03_2.9_4.7_05	7	11	14	15	4	0	21	3	8.9
i03_2.9_4.7_10	13	22	29	29	4	0	16	3	8.8
i03_3.2_2.0_05	7	14	14	14	4	0	5	1	160.3
i03_3.2_2.0_10	14	18	27	27	6	1	10	1	189.1

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i03_3.2_2.3_05	7	9	12	13	4	0	7	3	181.9
i03_3.2_2.3_10	16	20	28	30	4	0	7	3	178.9
i03_3.2_2.6_05	6	15	15	15	4	0	5	1	320.7
i03_3.2_2.6_10	15	29	29	29	4	0	5	1	318.9
i03_3.2_2.9_05	7	10	14	14	5	1	12	1	265.2
i03_3.2_2.9_10	15	23	31	31	4	0	13	3	250.8
i03_3.2_3.2_05	7	16	16	16	5	1	8	1	197.8
i03_3.2_3.2_10	16	32	32	32	8	2	7	1	329.4
i03_3.2_3.5_05	8	13	13	13	4	0	12	1	309.0
i03_3.2_3.5_10	14	17	25	26	7	2	19	1	time limit
i03_3.2_3.8_05	11	11	16	15	15	5	10	1	time limit
i03_3.2_3.8_10	22	22	32	32	15	5	19	1	time limit
i03_3.2_4.1_05	7	9	12	12	4	0	20	3	228.2
i03_3.2_4.1_10	15	21	28	27	4	0	14	3	231.3
i03_3.2_4.4_05	8	12	15	16	4	0	17	3	276.5
i03_3.2_4.4_10	15	19	26	26	4	0	13	3	276.6
i03_3.2_4.7_05	7	9	12	12	4	0	10	3	141.1
i03_3.2_4.7_10	16	21	30	32	5	1	26	1	152.1
i03_3.5_2.0_05	7	9	13	14	9	3	11	1	6052.1
i03_3.5_2.0_10	16	16	23	24	15	5	8	1	time limit
i03_3.5_2.3_05	8	15	15	15	4	0	7	1	3728.3
i03_3.5_2.3_10	14	20	29	30	5	1	10	1	3756.9
i03_3.5_2.6_05	7	17	17	17	5	1	7	1	3745.3
i03_3.5_2.6_10	13	39	39	39	4	0	5	1	3736.8
i03_3.5_2.9_05	9	16	16	17	8	2	13	3	time limit
i03_3.5_2.9_10	12	31	31	31	4	0	6	1	time limit
i03_3.5_3.2_05	7	18	18	18	4	0	5	1	time limit
i03_3.5_3.2_10	15	30	30	30	7	2	14	10	time limit
i03_3.5_3.5_05	8	9	14	15	5	1	12	2	4886.7
i03_3.5_3.5_10	14	19	27	27	5	1	23	1	5042.7
i03_3.5_3.8_05	6	13	13	13	4	0	5	1	time limit
i03_3.5_3.8_10	15	21	27	29	4	0	18	3	time limit
i03_3.5_4.1_05	6	9	12	12	4	0	22	3	3553.2
i03_3.5_4.1_10	14	19	24	25	4	0	21	3	3582.9
i03_3.5_4.4_05	8	9	14	15	12	4	29	1	time limit
i03_3.5_4.4_10	13	18	22	23	4	0	26	3	5203.6
i03_3.5_4.7_05	6	9	12	13	5	1	23	1	3311.8
i03_3.5_4.7_10	14	19	24	24	4	0	10	3	3042.6
i03_3.8_2.0_05	9	10	15	16	12	5	11	1	time limit
i03_3.8_2.0_10	14	34	34	34	6	2	5	1	time limit
i03_3.8_2.3_05	7	20	20	20	4	1	3	1	time limit
i03_3.8_2.3_10	15	33	34	34	6	2	11	1	time limit
i03_3.8_2.6_05	6	15	15	15	5	1	6	1	time limit
i03_3.8_2.6_10	12	26	26	26	5	1	6	1	time limit
i03_3.8_2.9_05	9	29	29	29	4	1	6	1	time limit
i03_3.8_2.9_10	13	36	36	36	6	2	6	1	time limit
i03_3.8_3.2_05	8	19	19	19	5	1	8	1	time limit
i03_3.8_3.2_10	14	25	25	25	18	8	5	1	time limit

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i03_3.8_3.5_05	8	11	16	16	6	2	15	1	time limit
i03_3.8_3.5_10	15	17	26	27	5	1	10	1	time limit
i03_3.8_3.8_05	11	11	14	14	26	14	17	9	time limit
i03_3.8_3.8_10	15	46	46	46	5	2	5	1	time limit
i03_3.8_4.1_05	8	9	13	14	5	1	21	1	time limit
i03_3.8_4.1_10	16	20	29	31	9	3	18	10	time limit
i03_3.8_4.4_05	6	9	13	13	12	5	14	1	time limit
i03_3.8_4.4_10	12	16	21	23	6	2	17	1	time limit
i03_3.8_4.7_05	7	9	11	11	5	1	23	1	time limit
i03_3.8_4.7_10	13	20	28	31	4	1	7	3	time limit
i03_4.1_2.0_05	7	8	11	11	20	9	9	1	time limit
i03_4.1_2.0_10	15	27	27	27	6	2	5	1	time limit
i03_4.1_2.3_05	6	14	14	14	6	2	7	1	time limit
i03_4.1_2.3_10	14	18	25	25	5	1	7	1	time limit
i03_4.1_2.6_05	6	24	24	24	5	2	3	1	time limit
i03_4.1_2.6_10	14	25	27	28	10	4	9	1	time limit
i03_4.1_2.9_05	9	16	17	16	6	2	13	1	time limit
i03_4.1_2.9_10	15	33	33	33	19	9	10	1	time limit
i03_4.1_3.2_05	7	18	18	18	6	2	8	1	time limit
i03_4.1_3.2_10	12	29	29	29	12	5	7	1	time limit
i03_4.1_3.5_05	9	10	14	15	5	1	12	1	time limit
i03_4.1_3.5_10	13	18	24	26	5	1	12	1	time limit
i03_4.1_3.8_05	8	15	15	15	20	9	7	1	time limit
i03_4.1_3.8_10	15	22	30	30	8	3	15	1	time limit
i03_4.1_4.1_05	10	10	14	14	29	17	19	1	time limit
i03_4.1_4.1_10	14	18	25	26	5	1	13	1	time limit
i03_4.1_4.4_05	7	9	12	13	6	2	19	1	time limit
i03_4.1_4.4_10	17	40	40	40	5	2	10	1	time limit
i03_4.1_4.7_05	8	11	15	15	6	2	22	1	time limit
i03_4.1_4.7_10	14	17	22	22	5	1	15	12	time limit
i03_4.4_2.0_05	8	13	14	15	6	2	6	1	time limit
i03_4.4_2.0_10	11	14	22	23	12	5	10	1	time limit
i03_4.4_2.3_05	8	18	18	18	5	1	7	1	time limit
i03_4.4_2.3_10	13	27	27	27	6	2	5	1	time limit
i03_4.4_2.6_05	8	9	13	13	18	8	10	1	time limit
i03_4.4_2.6_10	14	51	51	51	5	2	3	1	time limit
i03_4.4_2.9_05	6	25	25	25	5	2	3	1	time limit
i03_4.4_2.9_10	13	63	63	63	5	2	3	1	time limit
i03_4.4_3.2_05	7	14	14	14	6	2	8	1	time limit
i03_4.4_3.2_10	14	30	30	30	6	2	7	1	time limit
i03_4.4_3.5_05	8	17	17	17	5	2	15	2	time limit
i03_4.4_3.5_10	13	18	25	25	11	4	17	1	time limit
i03_4.4_3.8_05	8	16	16	16	5	2	6	1	time limit
i03_4.4_3.8_10	13	19	23	27	5	1	9	3	time limit
i03_4.4_4.1_05	7	9	13	13	6	2	14	1	time limit
i03_4.4_4.1_10	16	21	27	29	10	4	16	2	time limit
i03_4.4_4.4_05	7	8	11	11	5	1	16	3	time limit
i03_4.4_4.4_10	12	31	31	31	5	2	10	1	time limit

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i03_4.4_4.7_05	9	12	17	18	5	2	15	1	time limit
i03_4.4_4.7_10	16	16	23	24	26	14	20	1	time limit
i03_4.7_2.0_05	6	11	14	15	10	4	11	1	time limit
i03_4.7_2.0_10	14	16	25	26	18	8	7	1	time limit
i03_4.7_2.3_05	8	17	17	17	6	2	7	1	time limit
i03_4.7_2.3_10	16	29	29	29	6	2	10	1	time limit
i03_4.7_2.6_05	6	11	11	11	6	2	7	1	time limit
i03_4.7_2.6_10	12	34	34	34	6	2	5	1	time limit
i03_4.7_2.9_05	11	11	15	15	37	20	18	1	time limit
i03_4.7_2.9_10	12	31	31	31	12	5	5	1	time limit
i03_4.7_3.2_05	6	15	15	15	14	6	7	1	time limit
i03_4.7_3.2_10	16	60	60	60	5	2	5	1	time limit
i03_4.7_3.5_05	8	24	24	24	5	2	6	1	time limit
i03_4.7_3.5_10	15	21	28	28	8	3	23	1	time limit
i03_4.7_3.8_05	8	12	12	13	6	2	9	1	time limit
i03_4.7_3.8_10	13	16	27	28	6	2	13	1	time limit
i03_4.7_4.1_05	8	16	16	16	5	2	10	1	time limit
i03_4.7_4.1_10	13	17	24	24	10	4	17	1	time limit
i03_4.7_4.4_05	8	8	11	11	6	2	14	1	time limit
i03_4.7_4.4_10	12	14	21	21	6	2	11	1	time limit
i03_4.7_4.7_05	8	9	13	14	6	2	22	1	time limit
i03_4.7_4.7_10	16	20	25	26	5	1	15	2	time limit
i04_2.0_2.0_05	9	24	24	24	6	0	7	1	0.5
i04_2.0_2.0_10	25	50	50	50	7	0	9	1	1.0
i04_2.0_2.3_05	8	24	24	24	6	0	7	1	0.8
i04_2.0_2.3_10	14	36	36	37	7	0	8	1	0.7
i04_2.0_2.6_05	9	25	25	27	6	0	8	1	0.7
i04_2.0_2.6_10	14	33	33	32	7	0	11	1	0.6
i04_2.0_2.9_05	10	33	33	33	6	0	7	1	0.7
i04_2.0_2.9_10	18	21	33	33	7	0	17	3	0.7
i04_2.0_3.2_05	8	39	39	39	6	0	9	1	0.8
i04_2.0_3.2_10	17	44	44	44	6	0	14	1	0.5
i04_2.0_3.5_05	9	17	17	17	7	0	16	1	0.6
i04_2.0_3.5_10	17	23	33	34	7	0	23	3	0.6
i04_2.0_3.8_05	12	17	23	24	6	0	20	3	1.7
i04_2.0_3.8_10	15	24	35	36	6	0	21	3	0.5
i04_2.0_4.1_05	8	11	15	16	7	0	21	3	0.6
i04_2.0_4.1_10	16	22	30	31	7	0	20	3	0.5
i04_2.0_4.4_05	13	15	20	21	7	0	29	3	1.0
i04_2.0_4.4_10	14	21	30	30	6	0	30	3	0.5
i04_2.0_4.7_05	9	12	16	17	7	0	50	3	0.8
i04_2.0_4.7_10	36	39	55	58	6	0	18	3	1.9
i04_2.3_2.0_05	9	19	19	19	6	0	6	1	0.9
i04_2.3_2.0_10	26	30	47	50	9	0	9	3	2.8
i04_2.3_2.3_05	8	12	18	18	7	0	10	4	1.0
i04_2.3_2.3_10	19	30	47	45	7	0	9	3	1.1
i04_2.3_2.6_05	9	12	20	20	7	0	12	3	0.8
i04_2.3_2.6_10	14	22	33	34	7	0	20	3	1.1

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i04_2.3_2.9_05	10	36	36	36	6	0	6	1	1.0
i04_2.3_2.9_10	16	44	44	44	6	0	525	1	0.7
i04_2.3_3.2_05	8	38	38	32	5	0	6	1	1.3
i04_2.3_3.2_10	16	41	41	41	6	0	7	1	1.2
i04_2.3_3.5_05	9	24	24	24	6	0	14	1	1.1
i04_2.3_3.5_10	22	39	39	40	8	0	19	1	3.7
i04_2.3_3.8_05	8	15	20	23	6	0	22	6	0.8
i04_2.3_3.8_10	13	19	26	26	7	0	21	3	1.2
i04_2.3_4.1_05	8	11	15	15	7	0	28	3	2.0
i04_2.3_4.1_10	17	25	34	35	7	0	24	3	1.1
i04_2.3_4.4_05	12	14	19	19	10	0	14	3	3.8
i04_2.3_4.4_10	16	26	37	36	6	0	23	3	1.1
i04_2.3_4.7_05	8	14	19	19	6	0	29	3	0.9
i04_2.3_4.7_10	18	29	39	39	6	0	28	3	0.9
i04_2.6_2.0_05	11	23	23	23	11	1	11	1	814.8
i04_2.6_2.0_10	13	21	29	30	7	0	9	3	3.2
i04_2.6_2.3_05	8	18	18	18	8	0	11	1	5.1
i04_2.6_2.3_10	17	56	56	56	4	0	4	1	2.9
i04_2.6_2.6_05	8	11	16	16	9	0	8	3	5.7
i04_2.6_2.6_10	14	49	49	49	6	0	8	1	2.8
i04_2.6_2.9_05	8	13	17	17	9	0	11	3	5.4
i04_2.6_2.9_10	21	25	34	34	12	1	20	3	11.5
i04_2.6_3.2_05	8	30	30	30	6	0	6	1	2.7
i04_2.6_3.2_10	15	25	34	35	7	0	18	3	2.1
i04_2.6_3.5_05	10	10	15	23	11	1	20	3	2087.5
i04_2.6_3.5_10	14	35	35	35	6	0	8	1	3.2
i04_2.6_3.8_05	8	14	19	20	6	0	29	3	1.7
i04_2.6_3.8_10	27	46	46	46	12	2	19	2	15.2
i04_2.6_4.1_05	8	12	15	16	7	0	27	3	2.9
i04_2.6_4.1_10	14	21	29	32	9	0	15	3	7.8
i04_2.6_4.4_05	11	13	17	17	10	0	24	3	9.9
i04_2.6_4.4_10	14	21	29	29	7	0	21	3	2.0
i04_2.6_4.7_05	10	15	21	21	6	0	19	3	2.0
i04_2.6_4.7_10	16	21	32	31	8	0	25	3	6.9
i04_2.9_2.0_05	9	20	20	20	7	0	8	1	11.4
i04_2.9_2.0_10	16	30	30	31	9	1	11	1	28.6
i04_2.9_2.3_05	7	13	16	17	7	0	13	4	12.7
i04_2.9_2.3_10	16	26	37	38	8	1	26	3	14.3
i04_2.9_2.6_05	9	14	19	20	7	0	18	3	13.7
i04_2.9_2.6_10	15	30	36	36	7	0	11	3	20.8
i04_2.9_2.9_05	7	11	16	16	9	1	21	3	833.5
i04_2.9_2.9_10	18	28	38	39	10	1	13	8	828.9
i04_2.9_3.2_05	9	12	18	18	7	0	17	3	13.4
i04_2.9_3.2_10	17	26	36	37	7	0	23	3	13.2
i04_2.9_3.5_05	10	24	24	24	8	1	14	1	17.6
i04_2.9_3.5_10	18	25	33	35	7	0	21	3	14.5
i04_2.9_3.8_05	8	13	17	18	7	0	32	3	10.2
i04_2.9_3.8_10	16	25	33	34	9	1	17	3	22.3

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i04_2.9_4.1_05	7	16	16	16	6	0	13	1	11.2
i04_2.9_4.1_10	14	28	28	29	12	2	21	1	132.0
i04_2.9_4.4_05	8	13	17	17	7	0	34	3	12.9
i04_2.9_4.4_10	18	24	31	31	7	0	36	3	12.4
i04_2.9_4.7_05	9	12	16	16	7	0	40	3	11.0
i04_2.9_4.7_10	17	26	34	35	7	0	29	9	10.6
i04_3.2_2.0_05	10	13	19	20	11	2	12	1	687.1
i04_3.2_2.0_10	19	30	36	38	14	4	8	1	3399.2
i04_3.2_2.3_05	11	14	19	20	21	6	11	1	time limit
i04_3.2_2.3_10	14	38	38	38	9	1	13	1	200.7
i04_3.2_2.6_05	10	24	24	24	6	0	6	1	356.4
i04_3.2_2.6_10	17	35	52	53	8	2	8	1	336.0
i04_3.2_2.9_05	9	32	32	32	6	0	9	1	280.8
i04_3.2_2.9_10	18	33	33	33	18	5	16	1	1902.6
i04_3.2_3.2_05	12	14	20	21	28	12	31	1	time limit
i04_3.2_3.2_10	19	37	38	41	8	1	13	3	202.2
i04_3.2_3.5_05	8	12	17	18	11	2	20	1	442.5
i04_3.2_3.5_10	17	24	35	36	8	1	7	3	316.4
i04_3.2_3.8_05	8	12	18	19	6	0	12	3	time limit
i04_3.2_3.8_10	13	18	23	24	16	4	13	3	time limit
i04_3.2_4.1_05	7	11	14	14	7	0	27	3	229.9
i04_3.2_4.1_10	15	25	34	37	15	4	13	3	time limit
i04_3.2_4.4_05	7	21	21	20	5	0	10	1	333.5
i04_3.2_4.4_10	16	26	39	39	8	2	46	1	285.0
i04_3.2_4.7_05	8	13	17	18	8	1	23	1	147.9
i04_3.2_4.7_10	16	24	31	32	7	0	23	3	144.2
i04_3.5_2.0_05	10	14	19	19	6	0	6	3	5189.0
i04_3.5_2.0_10	16	23	35	36	26	10	13	1	time limit
i04_3.5_2.3_05	11	11	17	17	26	8	17	1	time limit
i04_3.5_2.3_10	14	34	34	34	6	0	6	1	3724.2
i04_3.5_2.6_05	8	18	18	19	7	0	9	1	3596.2
i04_3.5_2.6_10	27	27	39	41	27	10	15	1	time limit
i04_3.5_2.9_05	10	13	19	19	9	2	23	1	time limit
i04_3.5_2.9_10	18	24	31	31	17	4	13	1	time limit
i04_3.5_3.2_05	9	14	18	18	8	1	24	3	time limit
i04_3.5_3.2_10	16	34	34	34	8	1	6	1	time limit
i04_3.5_3.5_05	7	17	17	17	13	4	14	1	time limit
i04_3.5_3.5_10	27	27	38	47	87	48	18	1	time limit
i04_3.5_3.8_05	8	11	15	16	14	4	21	1	time limit
i04_3.5_3.8_10	14	22	29	29	10	2	26	1	time limit
i04_3.5_4.1_05	8	13	18	19	7	1	30	1	3703.7
i04_3.5_4.1_10	16	24	33	33	7	1	33	1	3775.6
i04_3.5_4.4_05	8	11	14	15	7	0	40	3	5490.4
i04_3.5_4.4_10	17	24	30	31	7	0	32	3	5245.5
i04_3.5_4.7_05	8	13	22	21	7	1	13	1	3241.2
i04_3.5_4.7_10	15	21	26	26	7	0	10	3	3024.1
i04_3.8_2.0_05	7	17	17	17	12	4	13	1	time limit
i04_3.8_2.0_10	16	36	36	36	7	1	6	1	time limit

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i04_3.8_2.3_05	9	10	14	18	29	12	19	1	time limit
i04_3.8_2.3_10	16	39	39	39	17	7	12	1	time limit
i04_3.8_2.6_05	12	14	20	21	37	18	27	1	time limit
i04_3.8_2.6_10	19	46	46	46	7	1	6	1	time limit
i04_3.8_2.9_05	9	42	42	42	6	2	4	1	time limit
i04_3.8_2.9_10	18	54	54	54	7	1	7	1	time limit
i04_3.8_3.2_05	9	36	36	36	8	2	8	1	time limit
i04_3.8_3.2_10	17	71	71	71	8	2	10	1	time limit
i04_3.8_3.5_05	8	11	14	14	8	1	24	3	time limit
i04_3.8_3.5_10	16	38	38	38	22	10	9	1	time limit
i04_3.8_3.8_05	8	11	15	17	8	1	24	3	time limit
i04_3.8_3.8_10	18	55	55	55	7	3	9	1	time limit
i04_3.8_4.1_05	7	11	14	15	22	9	17	1	time limit
i04_3.8_4.1_10	18	26	36	37	14	5	21	48	time limit
i04_3.8_4.4_05	8	13	17	17	30	14	32	1	time limit
i04_3.8_4.4_10	14	19	27	27	18	6	21	1	time limit
i04_3.8_4.7_05	7	8	10	14	31	13	38	3	time limit
i04_3.8_4.7_10	17	24	32	32	9	2	26	1	time limit
i04_4.1_2.0_05	10	13	20	19	17	6	11	1	time limit
i04_4.1_2.0_10	18	38	40	40	9	3	7	1	time limit
i04_4.1_2.3_05	8	15	15	16	10	3	14	1	time limit
i04_4.1_2.3_10	17	50	50	50	7	1	4	1	time limit
i04_4.1_2.6_05	8	19	19	19	8	2	10	1	time limit
i04_4.1_2.6_10	18	48	48	48	9	3	7	1	time limit
i04_4.1_2.9_05	9	18	18	18	9	2	8	1	time limit
i04_4.1_2.9_10	25	26	35	35	43	21	16	1	time limit
i04_4.1_3.2_05	8	18	18	19	19	7	8	1	time limit
i04_4.1_3.2_10	18	40	40	40	9	2	11	1	time limit
i04_4.1_3.5_05	10	19	19	23	8	2	233	1	time limit
i04_4.1_3.5_10	18	35	36	36	12	5	15	1	time limit
i04_4.1_3.8_05	8	11	16	19	7	1	8	3	time limit
i04_4.1_3.8_10	18	25	32	32	9	2	20	1	time limit
i04_4.1_4.1_05	8	10	13	13	30	13	8	1	time limit
i04_4.1_4.1_10	16	35	35	35	8	2	24	2	time limit
i04_4.1_4.4_05	9	14	17	17	9	3	22	2	time limit
i04_4.1_4.4_10	18	19	27	29	28	12	28	7	time limit
i04_4.1_4.7_05	9	11	16	17	8	2	29	1	time limit
i04_4.1_4.7_10	15	23	34	37	7	1	11	3	time limit
i04_4.4_2.0_05	8	19	19	19	10	4	5	1	time limit
i04_4.4_2.0_10	18	18	27	27	45	23	16	1	time limit
i04_4.4_2.3_05	8	14	14	14	43	21	7	2	time limit
i04_4.4_2.3_10	23	25	36	37	53	29	12	1	time limit
i04_4.4_2.6_05	7	28	28	28	8	2	4	1	time limit
i04_4.4_2.6_10	22	25	35	35	157	96	15	5	time limit
i04_4.4_2.9_05	13	13	18	21	239	156	31	2	time limit
i04_4.4_2.9_10	19	43	43	43	11	4	11	1	time limit
i04_4.4_3.2_05	7	28	28	28	20	9	7	1	time limit
i04_4.4_3.2_10	18	41	41	41	12	4	13	1	time limit

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i04_4.4_3.5_05	11	11	16	21	256	166	19	1	time limit
i04_4.4_3.5_10	14	22	29	29	20	8	14	3	time limit
i04_4.4_3.8_05	7	24	24	24	30	14	10	1	time limit
i04_4.4_3.8_10	14	21	29	32	10	3	15	1	time limit
i04_4.4_4.1_05	8	13	18	18	10	4	14	2	time limit
i04_4.4_4.1_10	19	25	35	36	35	17	35	1	time limit
i04_4.4_4.4_05	7	15	15	15	10	4	22	1	time limit
i04_4.4_4.4_10	15	26	34	35	23	10	32	2	time limit
i04_4.4_4.7_05	10	13	18	18	15	6	25	1	time limit
i04_4.4_4.7_10	19	26	36	36	18	7	20	1	time limit
i04_4.7_2.0_05	11	11	16	16	254	163	8	1	time limit
i04_4.7_2.0_10	17	18	27	27	51	27	17	1	time limit
i04_4.7_2.3_05	8	13	17	18	19	8	23	1	time limit
i04_4.7_2.3_10	16	20	30	31	38	19	16	1	time limit
i04_4.7_2.6_05	7	23	23	23	10	4	7	1	time limit
i04_4.7_2.6_10	16	33	33	33	10	3	10	1	time limit
i04_4.7_2.9_05	8	17	17	17	17	7	17	1	time limit
i04_4.7_2.9_10	15	32	47	54	9	3	11	1	time limit
i04_4.7_3.2_05	9	16	16	17	10	3	11	1	time limit
i04_4.7_3.2_10	18	34	34	34	13	5	14	1	time limit
i04_4.7_3.5_05	8	15	22	22	9	3	16	1	time limit
i04_4.7_3.5_10	13	42	42	42	10	4	8	1	time limit
i04_4.7_3.8_05	9	12	17	17	20	8	18	1	time limit
i04_4.7_3.8_10	16	18	26	26	41	21	20	1	time limit
i04_4.7_4.1_05	11	11	16	15	54	27	22	1	time limit
i04_4.7_4.1_10	14	34	34	34	46	24	11	1	time limit
i04_4.7_4.4_05	9	11	14	15	9	2	18	1	time limit
i04_4.7_4.4_10	17	19	25	25	52	27	23	1	time limit
i04_4.7_4.7_05	10	16	21	21	21	9	45	2	time limit
i04_4.7_4.7_10	15	23	30	33	9	3	12	1	time limit
i05_2.0_2.0_05	10	24	24	24	10	0	8	1	0.6
i05_2.0_2.0_10	20	59	59	59	10	0	10	1	1.4
i05_2.0_2.3_05	10	15	26	26	10	0	8	3	0.8
i05_2.0_2.3_10	19	47	47	47	10	0	10	1	0.7
i05_2.0_2.6_05	11	31	31	32	9	0	12	1	0.8
i05_2.0_2.6_10	17	46	46	46	10	0	15	1	0.7
i05_2.0_2.9_05	9	33	33	33	9	0	10	1	0.7
i05_2.0_2.9_10	27	50	50	50	10	0	14	1	1.9
i05_2.0_3.2_05	10	26	26	26	9	0	15	1	0.6
i05_2.0_3.2_10	20	40	40	40	11	0	13	1	0.8
i05_2.0_3.5_05	9	13	19	19	10	0	31	3	0.9
i05_2.0_3.5_10	20	25	37	40	10	0	29	3	0.9
i05_2.0_3.8_05	14	15	22	23	11	0	18	3	1.1
i05_2.0_3.8_10	19	44	44	44	10	0	10	1	0.8
i05_2.0_4.1_05	9	15	20	21	8	0	30	5	0.9
i05_2.0_4.1_10	24	32	48	48	10	0	22	3	1.1
i05_2.0_4.4_05	9	12	18	19	10	0	27	3	0.7
i05_2.0_4.4_10	21	29	40	43	10	0	23	3	0.6

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i05_2.0_4.7_05	10	15	23	22	9	0	25	3	0.9
i05_2.0_4.7_10	17	25	34	36	10	0	22	3	0.8
i05_2.3_2.0_05	11	24	24	26	10	0	7	1	1.1
i05_2.3_2.0_10	27	38	53	58	12	0	14	3	2.5
i05_2.3_2.3_05	10	24	33	33	8	0	7	3	1.4
i05_2.3_2.3_10	18	61	61	61	10	0	11	1	1.4
i05_2.3_2.6_05	9	40	40	40	10	0	11	1	1.1
i05_2.3_2.6_10	27	34	52	55	13	0	16	3	3.2
i05_2.3_2.9_05	8	30	30	30	10	0	7	1	2.2
i05_2.3_2.9_10	24	61	61	61	12	0	21	1	9.7
i05_2.3_3.2_05	9	32	32	32	9	0	15	1	1.3
i05_2.3_3.2_10	29	34	47	50	14	0	29	3	5.2
i05_2.3_3.5_05	12	17	24	25	13	0	23	4	6.0
i05_2.3_3.5_10	17	23	32	34	12	0	18	3	2.0
i05_2.3_3.8_05	12	14	20	21	15	1	38	3	10.9
i05_2.3_3.8_10	33	36	52	55	17	0	28	3	10.7
i05_2.3_4.1_05	9	12	16	16	12	0	39	3	2.1
i05_2.3_4.1_10	16	27	37	39	10	0	19	3	1.5
i05_2.3_4.4_05	9	15	23	24	9	0	33	3	1.5
i05_2.3_4.4_10	17	34	51	51	8	0	12	3	1.4
i05_2.3_4.7_05	9	14	18	19	10	0	41	3	1.4
i05_2.3_4.7_10	17	26	34	36	10	0	25	3	1.1
i05_2.6_2.0_05	9	12	19	20	11	0	11	3	2.5
i05_2.6_2.0_10	18	28	41	41	10	0	7	3	2.1
i05_2.6_2.3_05	12	24	24	24	15	1	13	1	10.5
i05_2.6_2.3_10	22	45	54	56	10	0	13	3	4.5
i05_2.6_2.6_05	9	36	36	26	10	0	10	1	3.1
i05_2.6_2.6_10	19	49	49	49	10	0	12	1	3.0
i05_2.6_2.9_05	13	17	23	23	17	3	28	3	time limit
i05_2.6_2.9_10	18	58	58	58	10	0	9	1	3.4
i05_2.6_3.2_05	9	22	22	22	10	0	15	1	2.6
i05_2.6_3.2_10	16	76	76	76	14	0	13	1	15.6
i05_2.6_3.5_05	9	12	17	18	12	0	25	13	4.4
i05_2.6_3.5_10	21	29	40	41	11	0	26	6	3.5
i05_2.6_3.8_05	9	16	21	22	10	0	31	3	3.7
i05_2.6_3.8_10	16	21	29	29	12	0	19	3	6.3
i05_2.6_4.1_05	9	15	20	21	10	0	43	3	2.6
i05_2.6_4.1_10	17	33	45	46	8	0	27	3	5.0
i05_2.6_4.4_05	8	11	15	16	13	0	26	3	7.1
i05_2.6_4.4_10	21	30	43	45	10	0	24	3	4.0
i05_2.6_4.7_05	10	13	17	18	13	0	48	3	5.7
i05_2.6_4.7_10	21	31	41	41	10	0	33	3	2.7
i05_2.9_2.0_05	10	15	22	23	15	2	14	1	74.2
i05_2.9_2.0_10	26	29	48	49	24	6	11	1	3994.6
i05_2.9_2.3_05	8	27	27	27	8	0	5	1	19.2
i05_2.9_2.3_10	20	45	67	67	10	1	7	3	20.6
i05_2.9_2.6_05	10	27	27	27	10	0	14	1	16.7
i05_2.9_2.6_10	19	44	44	44	10	0	12	1	19.5

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i05_2.9_2.9_05	13	16	24	24	22	3	31	1	4983.5
i05_2.9_2.9_10	22	46	46	46	12	1	27	1	817.1
i05_2.9_3.2_05	8	18	18	19	13	1	18	1	23.5
i05_2.9_3.2_10	22	47	47	47	22	4	12	1	369.7
i05_2.9_3.5_05	12	14	20	19	17	3	36	1	3887.4
i05_2.9_3.5_10	27	31	43	45	19	3	28	1	2519.3
i05_2.9_3.8_05	8	15	20	21	9	0	26	3	14.5
i05_2.9_3.8_10	23	30	44	45	18	3	31	3	264.7
i05_2.9_4.1_05	9	15	21	22	10	0	26	3	15.4
i05_2.9_4.1_10	29	36	47	51	22	4	49	3	1080.0
i05_2.9_4.4_05	13	14	20	20	52	20	27	2	time limit
i05_2.9_4.4_10	20	32	44	44	10	0	28	3	15.2
i05_2.9_4.7_05	9	14	18	18	11	0	38	3	10.4
i05_2.9_4.7_10	15	25	32	32	10	0	29	3	10.6
i05_3.2_2.0_05	10	14	22	22	21	4	13	1	1379.4
i05_3.2_2.0_10	20	29	43	45	21	5	17	3	1102.0
i05_3.2_2.3_05	9	13	19	20	33	11	15	1	3045.5
i05_3.2_2.3_10	18	47	65	65	13	4	10	1	261.1
i05_3.2_2.6_05	9	19	24	26	18	4	12	4	time limit
i05_3.2_2.6_10	29	51	51	52	19	4	14	1	4556.8
i05_3.2_2.9_05	10	43	43	43	8	0	10	1	318.9
i05_3.2_2.9_10	16	39	39	39	19	4	10	1	378.9
i05_3.2_3.2_05	9	29	29	29	11	1	8	1	246.1
i05_3.2_3.2_10	18	38	38	38	10	0	10	1	211.9
i05_3.2_3.5_05	8	13	18	19	12	1	26	3	358.4
i05_3.2_3.5_10	18	30	44	51	10	0	19	3	325.0
i05_3.2_3.8_05	12	13	20	22	48	18	27	1	time limit
i05_3.2_3.8_10	14	23	31	34	10	0	32	3	time limit
i05_3.2_4.1_05	9	17	22	21	19	6	12	97	5083.6
i05_3.2_4.1_10	17	20	27	27	26	6	37	1	time limit
i05_3.2_4.4_05	10	17	25	25	7	0	14	3	540.1
i05_3.2_4.4_10	17	23	39	39	9	0	23	9	371.1
i05_3.2_4.7_05	10	11	17	19	10	0	32	3	158.5
i05_3.2_4.7_10	18	26	35	36	14	3	44	2	195.1
i05_3.5_2.0_05	13	14	22	24	117	67	12	1	time limit
i05_3.5_2.0_10	27	60	60	60	27	6	12	1	time limit
i05_3.5_2.3_05	9	17	19	20	55	26	13	2	time limit
i05_3.5_2.3_10	20	28	41	42	30	10	16	1	time limit
i05_3.5_2.6_05	13	17	22	22	27	7	27	1	time limit
i05_3.5_2.6_10	20	47	47	47	12	2	13	1	4237.1
i05_3.5_2.9_05	10	11	22	23	11	2	39	1	time limit
i05_3.5_2.9_10	16	30	40	41	15	3	22	1	time limit
i05_3.5_3.2_05	9	14	20	22	11	1	26	1	time limit
i05_3.5_3.2_10	19	96	96	96	35	13	6	1	time limit
i05_3.5_3.5_05	10	15	20	21	16	3	28	16	time limit
i05_3.5_3.5_10	15	60	60	60	13	4	11	3	5548.7
i05_3.5_3.8_05	11	18	20	21	22	6	28	11	time limit
i05_3.5_3.8_10	15	52	52	52	10	2	13	1	time limit

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i05_3.5_4.1_05	9	27	27	24	8	0	11	1	3572.4
i05_3.5_4.1_10	19	23	31	35	77	37	24	2	time limit
i05_3.5_4.4_05	9	16	22	23	29	9	28	3	time limit
i05_3.5_4.4_10	16	23	31	32	22	6	23	1	time limit
i05_3.5_4.7_05	9	12	18	19	26	7	19	14	time limit
i05_3.5_4.7_10	16	46	46	50	12	3	26	1	4915.6
i05_3.8_2.0_05	8	27	27	27	14	5	8	1	time limit
i05_3.8_2.0_10	18	55	55	55	11	2	5	1	time limit
i05_3.8_2.3_05	14	22	24	25	43	17	19	1	time limit
i05_3.8_2.3_10	16	38	38	38	16	4	11	1	time limit
i05_3.8_2.6_05	17	17	24	24	521	353	22	7	time limit
i05_3.8_2.6_10	16	52	52	52	12	2	11	1	time limit
i05_3.8_2.9_05	8	42	42	42	21	8	5	1	time limit
i05_3.8_2.9_10	20	33	42	44	21	7	24	3	time limit
i05_3.8_3.2_05	13	14	18	19	135	76	42	9	time limit
i05_3.8_3.2_10	18	41	41	41	14	3	20	1	time limit
i05_3.8_3.5_05	9	36	36	36	12	4	10	1	time limit
i05_3.8_3.5_10	16	40	40	45	11	2	13	1	time limit
i05_3.8_3.8_05	8	25	25	31	10	2	12	1	time limit
i05_3.8_3.8_10	18	27	36	37	16	4	33	3	time limit
i05_3.8_4.1_05	9	14	19	19	79	46	37	10	time limit
i05_3.8_4.1_10	18	26	34	35	28	9	22	3	time limit
i05_3.8_4.4_05	8	11	16	16	63	33	26	1	time limit
i05_3.8_4.4_10	16	23	39	38	18	6	25	1	time limit
i05_3.8_4.7_05	8	16	22	23	15	6	26	3	time limit
i05_3.8_4.7_10	18	24	39	42	33	13	23	5	time limit
i05_4.1_2.0_05	9	23	23	23	21	7	13	1	time limit
i05_4.1_2.0_10	19	39	41	43	14	4	12	1	time limit
i05_4.1_2.3_05	10	24	24	24	14	4	14	1	time limit
i05_4.1_2.3_10	22	52	52	54	16	6	13	1	time limit
i05_4.1_2.6_05	10	15	19	20	42	19	14	1	time limit
i05_4.1_2.6_10	26	45	45	45	65	32	10	1	time limit
i05_4.1_2.9_05	13	15	21	21	64	32	31	4	time limit
i05_4.1_2.9_10	17	51	51	51	24	8	13	1	time limit
i05_4.1_3.2_05	10	18	25	27	23	8	27	1	time limit
i05_4.1_3.2_10	16	61	61	61	14	5	11	1	time limit
i05_4.1_3.5_05	10	12	16	17	47	21	29	1	time limit
i05_4.1_3.5_10	19	27	35	36	33	12	32	2	time limit
i05_4.1_3.8_05	9	13	19	21	11	1	10	3	time limit
i05_4.1_3.8_10	17	38	38	40	11	2	10	1	time limit
i05_4.1_4.1_05	10	14	20	20	41	17	36	1	time limit
i05_4.1_4.1_10	25	31	43	43	56	28	42	1	time limit
i05_4.1_4.4_05	10	12	15	15	13	2	37	1	time limit
i05_4.1_4.4_10	16	26	37	39	13	4	26	3	time limit
i05_4.1_4.7_05	9	11	17	17	10	1	24	3	time limit
i05_4.1_4.7_10	19	34	45	45	41	19	28	1	time limit
i05_4.4_2.0_05	9	23	23	23	14	4	13	1	time limit
i05_4.4_2.0_10	16	37	37	37	44	20	14	1	time limit

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i05_4.4_2.3_05	9	19	19	19	13	2	10	1	time limit
i05_4.4_2.3_10	16	30	43	43	93	53	12	1	time limit
i05_4.4_2.6_05	9	23	23	23	38	15	11	1	time limit
i05_4.4_2.6_10	16	70	70	70	11	2	6	1	time limit
i05_4.4_2.9_05	10	27	27	27	23	9	14	1	time limit
i05_4.4_2.9_10	26	30	41	40	106	60	24	1	time limit
i05_4.4_3.2_05	8	25	25	25	28	12	10	1	time limit
i05_4.4_3.2_10	17	64	64	64	15	5	7	1	time limit
i05_4.4_3.5_05	9	21	21	21	14	4	13	1	time limit
i05_4.4_3.5_10	18	43	44	43	14	4	12	3	time limit
i05_4.4_3.8_05	9	21	21	23	129	75	14	1	time limit
i05_4.4_3.8_10	18	28	41	43	30	12	36	1	time limit
i05_4.4_4.1_05	8	13	18	18	20	7	23	1	time limit
i05_4.4_4.1_10	21	33	48	49	15	6	29	1	time limit
i05_4.4_4.4_05	9	14	19	20	16	4	33	1	time limit
i05_4.4_4.4_10	16	25	36	37	23	9	52	2	time limit
i05_4.4_4.7_05	11	16	20	21	13	3	32	6	time limit
i05_4.4_4.7_10	18	21	37	39	16	6	25	1	time limit
i05_4.7_2.0_05	8	10	15	15	29	11	12	1	time limit
i05_4.7_2.0_10	18	27	39	39	16	4	17	1	time limit
i05_4.7_2.3_05	9	33	33	33	14	6	5	1	time limit
i05_4.7_2.3_10	20	33	35	38	190	116	14	3	time limit
i05_4.7_2.6_05	9	12	16	17	152	94	30	1	time limit
i05_4.7_2.6_10	25	28	38	41	446	291	28	1	time limit
i05_4.7_2.9_05	9	18	18	18	25	9	15	1	time limit
i05_4.7_2.9_10	21	67	67	67	59	31	13	1	time limit
i05_4.7_3.2_05	8	33	33	33	87	50	7	1	time limit
i05_4.7_3.2_10	17	26	39	40	20	7	40	2	time limit
i05_4.7_3.5_05	10	15	23	23	139	87	22	2	time limit
i05_4.7_3.5_10	19	28	41	44	18	6	47	25	time limit
i05_4.7_3.8_05	8	13	17	18	31	13	45	3	time limit
i05_4.7_3.8_10	16	28	39	43	21	8	33	2	time limit
i05_4.7_4.1_05	10	13	19	20	20	8	36	2	time limit
i05_4.7_4.1_10	18	26	34	35	35	14	38	1	time limit
i05_4.7_4.4_05	8	12	16	16	19	6	57	32	time limit
i05_4.7_4.4_10	20	27	38	38	14	4	48	1	time limit
i05_4.7_4.7_05	9	15	20	20	17	6	40	4	time limit
i05_4.7_4.7_10	29	31	46	44	140	88	46	32	time limit
i10_2.0_2.0_05	15	38	38	38	37	0	10	1	1.3
i10_2.0_2.0_10	37	73	73	73	40	0	15	1	2.9
i10_2.0_2.3_05	16	65	65	65	36	0	10	1	1.5
i10_2.0_2.3_10	37	94	94	95	40	0	20	1	2.7
i10_2.0_2.6_05	22	28	48	50	39	0	28	3	5.8
i10_2.0_2.6_10	26	87	87	87	38	0	15	1	2.4
i10_2.0_2.9_05	15	57	57	57	36	0	20	1	2.3
i10_2.0_2.9_10	45	127	127	127	38	0	22	1	5.2
i10_2.0_3.2_05	15	82	82	82	30	0	12	1	1.7
i10_2.0_3.2_10	34	101	101	101	35	0	26	1	4.3

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i10_2.0_3.5_05	15	20	29	29	43	0	28	3	2.7
i10_2.0_3.5_10	47	58	89	91	38	0	39	3	7.2
i10_2.0_3.8_05	20	23	35	36	41	0	39	3	4.5
i10_2.0_3.8_10	36	39	57	58	40	0	45	7	2.6
i10_2.0_4.1_05	17	22	31	31	40	0	58	3	4.7
i10_2.0_4.1_10	29	48	71	74	38	0	27	17	1.3
i10_2.0_4.4_05	18	21	30	31	40	0	39	3	2.5
i10_2.0_4.4_10	28	34	59	58	38	0	37	3	1.5
i10_2.0_4.7_05	18	21	32	34	39	0	50	3	2.2
i10_2.0_4.7_10	43	55	81	89	33	0	53	3	9.7
i10_2.3_2.0_05	18	48	48	48	42	0	13	1	15.4
i10_2.3_2.0_10	25	63	63	63	37	0	10	1	2.8
i10_2.3_2.3_05	14	57	57	57	37	0	14	1	3.8
i10_2.3_2.3_10	47	62	101	101	46	0	19	3	18.6
i10_2.3_2.6_05	14	53	53	53	40	0	18	1	5.2
i10_2.3_2.6_10	28	100	100	100	37	0	18	1	4.2
i10_2.3_2.9_05	18	44	44	44	45	0	24	1	12.2
i10_2.3_2.9_10	26	132	132	132	33	0	13	1	5.4
i10_2.3_3.2_05	19	54	54	54	45	0	26	1	12.3
i10_2.3_3.2_10	35	41	62	66	48	0	51	3	21.2
i10_2.3_3.5_05	17	19	28	30	41	0	39	3	4.3
i10_2.3_3.5_10	35	59	85	85	44	0	36	4	13.2
i10_2.3_3.8_05	13	28	41	42	33	0	41	3	3.9
i10_2.3_3.8_10	47	52	76	92	66	0	50	3	44.9
i10_2.3_4.1_05	17	21	29	31	43	0	35	11	6.6
i10_2.3_4.1_10	27	50	72	76	37	0	35	3	3.1
i10_2.3_4.4_05	15	23	34	35	39	0	56	3	10.3
i10_2.3_4.4_10	26	37	58	60	40	0	48	3	4.6
i10_2.3_4.7_05	14	23	38	37	36	0	39	17	3.0
i10_2.3_4.7_10	35	39	58	64	48	0	50	7	20.5
i10_2.6_2.0_05	18	36	36	38	69	3	27	1	time limit
i10_2.6_2.0_10	28	58	73	73	43	0	16	3	12.8
i10_2.6_2.3_05	15	45	45	45	29	0	14	1	7.1
i10_2.6_2.3_10	29	93	93	117	32	0	10	1	14.7
i10_2.6_2.6_05	18	25	38	40	102	11	26	3	time limit
i10_2.6_2.6_10	28	120	120	120	36	0	14	1	9.2
i10_2.6_2.9_05	17	24	34	36	49	1	38	3	703.8
i10_2.6_2.9_10	27	99	99	99	43	0	24	1	65.9
i10_2.6_3.2_05	14	26	39	42	36	0	36	3	8.3
i10_2.6_3.2_10	37	76	76	80	37	1	48	1	12.2
i10_2.6_3.5_05	20	37	55	57	35	1	27	7	23.1
i10_2.6_3.5_10	28	92	92	92	44	1	29	1	4337.2
i10_2.6_3.8_05	15	24	32	39	45	0	33	3	19.3
i10_2.6_3.8_10	29	43	58	59	40	0	46	7	9.7
i10_2.6_4.1_05	26	26	40	45	409	266	85	38	time limit
i10_2.6_4.1_10	30	57	82	84	35	0	43	3	13.8
i10_2.6_4.4_05	20	39	39	39	148	14	38	1	time limit
i10_2.6_4.4_10	26	45	62	64	42	0	47	3	18.1

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i10_2.6_4.7_05	14	25	37	38	36	0	33	6	7.8
i10_2.6_4.7_10	33	39	57	61	55	1	61	3	68.5
i10_2.9_2.0_05	20	20	33	36	591	437	28	4	time limit
i10_2.9_2.0_10	27	71	71	71	41	1	16	1	34.6
i10_2.9_2.3_05	14	55	55	55	40	1	16	1	20.5
i10_2.9_2.3_10	26	68	68	83	43	1	26	1	34.0
i10_2.9_2.6_05	18	54	54	54	83	17	29	1	time limit
i10_2.9_2.6_10	27	48	73	79	99	15	26	3	435.3
i10_2.9_2.9_05	19	24	34	36	81	14	31	10	5081.9
i10_2.9_2.9_10	36	89	89	89	58	5	29	1	1103.7
i10_2.9_3.2_05	13	32	32	34	44	2	39	1	31.3
i10_2.9_3.2_10	25	41	64	71	49	3	33	3	88.7
i10_2.9_3.5_05	15	21	32	33	38	0	56	3	16.3
i10_2.9_3.5_10	30	37	54	61	48	2	63	3	50.3
i10_2.9_3.8_05	22	26	35	37	197	94	50	125	time limit
i10_2.9_3.8_10	35	49	68	72	90	16	48	3	526.4
i10_2.9_4.1_05	15	23	32	32	44	2	44	3	36.0
i10_2.9_4.1_10	29	39	58	71	59	5	49	3	229.0
i10_2.9_4.4_05	13	21	29	30	78	12	48	3	3472.1
i10_2.9_4.4_10	27	45	68	74	39	2	63	3	42.2
i10_2.9_4.7_05	15	24	34	36	38	0	55	25	24.8
i10_2.9_4.7_10	25	42	60	64	40	0	47	12	58.3
i10_3.2_2.0_05	13	36	36	36	54	10	13	1	667.3
i10_3.2_2.0_10	30	64	65	69	80	20	18	1	time limit
i10_3.2_2.3_05	12	24	35	37	250	157	34	3	time limit
i10_3.2_2.3_10	26	126	126	126	39	1	10	1	194.4
i10_3.2_2.6_05	15	26	41	41	530	400	19	3	time limit
i10_3.2_2.6_10	32	52	61	70	89	23	37	1	time limit
i10_3.2_2.9_05	14	20	29	32	79	19	45	4	708.9
i10_3.2_2.9_10	31	66	66	66	62	11	31	1	1282.6
i10_3.2_3.2_05	14	24	36	38	74	15	27	3	2195.9
i10_3.2_3.2_10	28	40	62	67	65	13	65	3	time limit
i10_3.2_3.5_05	14	23	33	34	80	20	63	2	2882.0
i10_3.2_3.5_10	28	42	58	64	99	26	28	3	time limit
i10_3.2_3.8_05	15	20	27	29	62	8	54	40	time limit
i10_3.2_3.8_10	33	39	59	61	232	152	39	4	time limit
i10_3.2_4.1_05	13	20	31	30	99	28	38	3	4442.3
i10_3.2_4.1_10	28	35	55	55	45	3	39	9	358.9
i10_3.2_4.4_05	15	19	28	29	107	35	56	45	time limit
i10_3.2_4.4_10	33	33	59	63	180	93	77	13	time limit
i10_3.2_4.7_05	13	19	28	30	72	14	45	3	time limit
i10_3.2_4.7_10	34	35	66	67	619	457	77	98	time limit
i10_3.5_2.0_05	22	23	38	41	1475	1075	36	5	time limit
i10_3.5_2.0_10	29	96	96	96	76	16	10	4	time limit
i10_3.5_2.3_05	23	46	46	46	727	502	28	15	time limit
i10_3.5_2.3_10	28	123	123	123	34	0	10	1	time limit
i10_3.5_2.6_05	13	36	36	39	55	11	23	1	time limit
i10_3.5_2.6_10	26	56	56	56	567	407	22	1	time limit

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i10_3.5_2.9_05	13	55	55	55	45	4	17	1	time limit
i10_3.5_2.9_10	26	89	89	89	264	164	20	1	time limit
i10_3.5_3.2_05	15	51	51	51	39	5	16	1	time limit
i10_3.5_3.2_10	28	40	63	66	63	12	46	13	time limit
i10_3.5_3.5_05	15	42	42	42	386	253	26	8	time limit
i10_3.5_3.5_10	32	52	56	58	661	479	33	1	time limit
i10_3.5_3.8_05	13	32	32	32	81	21	31	2	time limit
i10_3.5_3.8_10	29	47	70	73	37	2	32	5	time limit
i10_3.5_4.1_05	19	25	32	33	106	31	51	12	time limit
i10_3.5_4.1_10	30	39	60	60	79	22	55	42	time limit
i10_3.5_4.4_05	15	25	36	36	48	7	72	92	time limit
i10_3.5_4.4_10	28	43	66	68	315	203	50	207	time limit
i10_3.5_4.7_05	17	19	31	32	452	284	73	50	time limit
i10_3.5_4.7_10	26	40	58	62	108	33	45	3	time limit
i10_3.8_2.0_05	22	22	34	36	3079	2293	33	2	time limit
i10_3.8_2.0_10	39	87	87	89	364	216	10	2	time limit
i10_3.8_2.3_05	15	20	43	41	181	113	17	1	time limit
i10_3.8_2.3_10	25	78	78	78	71	20	15	1	time limit
i10_3.8_2.6_05	13	31	32	33	251	166	34	3	time limit
i10_3.8_2.6_10	39	53	79	85	214	137	31	6	time limit
i10_3.8_2.9_05	16	19	29	31	601	436	35	2	time limit
i10_3.8_2.9_10	39	96	96	96	1216	900	26	1	time limit
i10_3.8_3.2_05	14	21	31	33	48	8	31	3	time limit
i10_3.8_3.2_10	39	46	78	82	811	630	66	3	time limit
i10_3.8_3.5_05	13	34	34	34	617	415	25	2	time limit
i10_3.8_3.5_10	34	53	79	98	176	94	20	3	time limit
i10_3.8_3.8_05	18	21	32	39	947	689	47	23	time limit
i10_3.8_3.8_10	34	38	59	65	879	646	54	6	time limit
i10_3.8_4.1_05	13	18	27	29	677	497	48	32	time limit
i10_3.8_4.1_10	29	44	68	67	100	46	41	25	time limit
i10_3.8_4.4_05	13	21	36	34	408	270	42	2	time limit
i10_3.8_4.4_10	29	34	51	53	1174	909	50	3	time limit
i10_3.8_4.7_05	13	21	36	36	193	110	52	2	time limit
i10_3.8_4.7_10	28	41	63	68	193	118	54	6	time limit
i10_4.1_2.0_05	14	41	41	41	160	97	26	2	time limit
i10_4.1_2.0_10	25	39	62	67	1333	1020	25	52	time limit
i10_4.1_2.3_05	14	56	56	45	146	88	12	1	time limit
i10_4.1_2.3_10	27	91	91	91	73	21	21	1	time limit
i10_4.1_2.6_05	13	31	54	54	121	64	18	1	time limit
i10_4.1_2.6_10	25	60	60	60	336	225	16	1	time limit
i10_4.1_2.9_05	15	33	33	33	57	12	30	1	time limit
i10_4.1_2.9_10	31	91	91	91	128	62	27	1	time limit
i10_4.1_3.2_05	13	15	24	26	382	270	67	2	time limit
i10_4.1_3.2_10	26	67	72	77	1350	992	56	11	time limit
i10_4.1_3.5_05	15	36	37	39	153	83	41	3	time limit
i10_4.1_3.5_10	28	45	61	62	76	22	58	1	time limit
i10_4.1_3.8_05	20	21	37	39	305	195	70	9	time limit
i10_4.1_3.8_10	36	44	69	71	255	160	67	4	time limit

instance	vol	dual	primal	GRASP	nfeas	ncands	nrp	nodes	enum time
i10_4.1_4.1_05	15	19	29	32	163	91	49	12	time limit
i10_4.1_4.1_10	27	54	90	92	322	212	23	52	time limit
i10_4.1_4.4_05	18	18	30	32	673	439	47	30	time limit
i10_4.1_4.4_10	27	34	55	57	205	112	61	10	time limit
i10_4.1_4.7_05	15	23	32	32	123	55	72	2	time limit
i10_4.1_4.7_10	28	55	80	85	64	19	41	8	time limit
i10_4.4_2.0_05	13	20	31	33	1128	856	23	3	time limit
i10_4.4_2.0_10	31	48	82	87	1222	958	32	2	time limit
i10_4.4_2.3_05	18	31	33	34	544	358	44	1	time limit
i10_4.4_2.3_10	27	68	68	68	247	158	26	1	time limit
i10_4.4_2.6_05	13	22	39	40	172	104	27	1	time limit
i10_4.4_2.6_10	29	66	70	70	459	321	17	1	time limit
i10_4.4_2.9_05	15	45	45	51	103	60	27	1	time limit
i10_4.4_2.9_10	26	77	77	77	433	295	16	1	time limit
i10_4.4_3.2_05	14	35	35	35	75	23	22	1	time limit
i10_4.4_3.2_10	39	69	69	69	322	188	27	1	time limit
i10_4.4_3.5_05	17	20	31	33	14181	11455	49	3	time limit
i10_4.4_3.5_10	28	76	76	76	602	404	107	2	time limit
i10_4.4_3.8_05	13	21	32	33	438	311	61	3	time limit
i10_4.4_3.8_10	26	41	72	77	420	287	62	2	time limit
i10_4.4_4.1_05	14	19	29	31	249	169	83	120	time limit
i10_4.4_4.1_10	28	41	62	61	56	17	27	1	time limit
i10_4.4_4.4_05	19	21	33	41	1879	1452	99	71	time limit
i10_4.4_4.4_10	28	51	73	75	75	32	45	9	time limit
i10_4.4_4.7_05	14	18	29	30	309	216	70	9	time limit
i10_4.4_4.7_10	30	44	60	63	70	23	58	16	time limit
i10_4.7_2.0_05	16	40	41	41	110	58	22	1	time limit
i10_4.7_2.0_10	27	103	103	103	334	222	10	1	time limit
i10_4.7_2.3_05	15	47	47	48	288	189	20	1	time limit
i10_4.7_2.3_10	36	76	76	76	428	282	18	1	time limit
i10_4.7_2.6_05	15	26	39	39	214	136	31	3	time limit
i10_4.7_2.6_10	27	63	63	64	676	476	18	1	time limit
i10_4.7_2.9_05	20	21	37	39	559	383	74	3	time limit
i10_4.7_2.9_10	26	65	65	72	66	18	34	1	time limit
i10_4.7_3.2_05	18	31	34	39	2181	1555	44	2	time limit
i10_4.7_3.2_10	29	65	75	78	192	128	57	2	time limit
i10_4.7_3.5_05	13	19	31	31	1734	1326	51	20	time limit
i10_4.7_3.5_10	27	28	50	54	603	412	52	10	time limit
i10_4.7_3.8_05	13	19	28	32	82	38	46	8	time limit
i10_4.7_3.8_10	29	37	56	62	224	135	36	3	time limit
i10_4.7_4.1_05	20	20	32	34	3538	2686	55	3	time limit
i10_4.7_4.1_10	27	42	64	68	807	617	48	3	time limit
i10_4.7_4.4_05	15	26	38	40	61	17	57	2	time limit
i10_4.7_4.4_10	24	37	57	59	267	186	83	77	time limit
i10_4.7_4.7_05	14	22	32	34	412	259	57	123	time limit
i10_4.7_4.7_10	28	42	69	78	782	578	87	3	time limit