# Algorithms for the circle packing problem based on mixed-integer DC programming

Satoru Masuda[*] Takayuki Okuno[†] Yoshiko Ikebe[‡]

**Abstract**

Circle packing problems are a class of packing problems which attempt to pack a given set of circles into a container with no overlap. In this paper, we focus on the circle packing problem proposed by López et.al. The problem is to pack circles of unequal size into a fixed size circular container, so as to maximize the total area of the packed circles. López et al. formulated this problem as a mixed-integer nonconvex quadratic programming problem, and proposed a heuristic method based on its continuous relaxation, by which they were able to solve instances with up to 40 circles.

In this paper, we propose an algorithm using mixed-integer DC programming. A DC program is an optimization problem in which the objective function can be represented by the difference of two convex functions, and a mixed-integer DC program is a DC program where some of the variables are restricted to integer values. By our method, we were able to obtain good solutions for problems with up to 60 circles.

## 1 Introduction

Packing problems, which are optimization problems in which a given set of objects are to be placed without overlap into a given container, are a class of well studied problems with many variations and applications[1, 3]. Almost all variations are known to be NP-hard, and most studies focus on the construction of efficient heuristic methods.

In this paper, we consider a problem concerning the packing of a given set of circles into a circular container. Among the many variations, we focus on the problem formulated by López et al. [2]. In this problem, we have a set of $n$ circles

[*]Tokyo University of Science
[†]RIKEN. Center for Advanced Intelligence Project
[‡]Tokyo University of Science

with radii $R_1, R_2, \ldots, R_n$, and a circular container of radius $R_0$. The objective is to choose and place the circles so that (i) there is no overlap, and (ii) the total area of the chosen circles is maximized.

López et al. [2] formulated this problem as a nonconvex quadratic mixed integer problem, and by using a commercial solver, obtained exact solutions for problems with few circles. They further proposed a heuristic method which successively solves a series of continuous relaxations, and reported that feasible solutions were obtained for instances with up to 40 circles.

In this paper, we propose a heuristic algorithm based on mixed integer DC programming. A DC program is an optimization problem having an objective function expressed as the difference of two convex functions, and a mixed integer DC program is a DC program in which some of the variables are restricted to integer values. Stationary points of mixed integer DC programs can be efficiently found by the algorithm proposed by Okuno et al. [5]. By applying our algorithm, we successfully obtained good solutions for instances with up to 60 circles.

This paper is organized as follows. In the next section, we describe the formulation and algorithm of López et al.[2], and in Section 3, we provide some fundamentals on mixed integer DC programming. In Section 4, we describe our heuristic algorithm, and in Section 5, we show the results of numerical experiments.

## 2 Formulation and algorithm of López et al.

Recall that we have $n$ circles of radius $R_1, R_2, \ldots, R_n$, which we wish to pack into a container of radius $R_0$, so as to maximize the total area.

In the formulation of [2], the coordinates of the center of the container are fixed to the origin $(0, 0)$. For each circle $i$ $(i = 1, 2, \ldots, n)$, we associate the variables $(x_i, y_i) \in \mathbb{R}^2$ corresponding to its center, and $\alpha_i \in \{0, 1\}$ expressing whether or not it is chosen .

By using these variables, the problem can be formulated as follows.

$$
\begin{array}{lll}
\underset{\boldsymbol{x},\boldsymbol{y},\boldsymbol{\alpha}}{\text{minimize}} & f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\alpha}) = -\pi \displaystyle\sum_{i=1}^{n} \alpha_i R_i^2 & \text{(F1)} \\
\text{subject to} & \alpha_i \alpha_j (R_i + R_j)^2 \leq (x_i - x_j)^2 + (y_i - y_j)^2 \quad (1 \leq i < j \leq n) & \text{(F2)} \\
& x_i^2 + y_i^2 \leq \alpha_i (R_0 - R_i)^2 \quad (i = 1, 2, \ldots, n) & \text{(F3)} \\
& -\alpha_i (R_0 - R_i) \leq x_i \leq \alpha_i (R_0 - R_i) \quad (i = 1, 2, \ldots, n) & \text{(F4)} \\
& -\alpha_i (R_0 - R_i) \leq y_i \leq \alpha_i (R_0 - R_i) \quad (i = 1, 2, \ldots, n) & \text{(F5)} \\
& \alpha_i \in \{0, 1\} \quad (i = 1, 2, \ldots, n) & \text{(F6)}
\end{array}
$$

We call this formulation (CPP1). In the above formulation, inequality (F2) forces

the constraint that the circles have no overlap, and inequality (F3) says that the chosen circles do not protrude from the container, and rejected circles are centered at the origin. Inequlities (F4) and (F5) are not strictly necessary, but provide upper and lower bounds on the coordinates of the centers, which can help when a solver is used.

López et al. [2] applied a commercial nonlinear optmization solver to the problem (CPP1), and reported that solutions were obtainable only for instances with $n$ in the vicinity of 10. Thus, they proposed a heuristic method based on a continuous relaxation.

Simply relaxing the constraint $\alpha_i \in \{0, 1\}$ to $0 \leq \alpha_i \leq 1$ will not result in good solutions, hence, they introduced an auxiliary variable $\delta > 0$, and proposed the following relaxation which replaces the $0-1$ constraint (F6) by the following two constraints. We will call the resulting problem ($\mathrm{CPP1_R}(\delta)$)

$$\begin{aligned} & \alpha_i \in \{0, 1\} \ \ (i = 1, 2, \ldots, n) \\ \Longrightarrow & \sum_{i=1}^{n} \alpha_i (1 - \alpha_i) \leq \delta, \qquad 0 \leq \alpha_i \leq 1 \ \ (i = 1, 2, \ldots, n). \end{aligned}$$

Note that when $\delta = 0$, this problem is equivalent to (CPP1). The heuristic method proposed in [2], initially sets $\delta = 0.05$, then solves a series of problems ($\mathrm{CPP1_R}(\delta)$) while reducing the value of $\delta$ by a fixed factor $\beta$. To assure that the computation time is not overly costly, they set a time limit ($10n$ sec.) to the nonlinear optimization solver. With this method they were able to obtain good solutions for instances with up to $n = 40$ circles. Their method is explicitly described in Algorithm 1.

## 2.1 Continuous DC programming

In this section, we briefly describe important properties of DC programming.

**Definition 2.1.** A real valued function $f : \mathbb{R}^n \to \mathbb{R}$ is a *DC function* (Difference of Convex function), if there exist two convex functions $g, h : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ satisfying

$$f(\boldsymbol{x}) = g(\boldsymbol{x}) - h(\boldsymbol{x}). \tag{1}$$

The above expression is called a *DC decomposition* of $f$. Since $g(\boldsymbol{x}) - h(\boldsymbol{x}) = (g(\boldsymbol{x}) + \varphi(\boldsymbol{x})) - (h(\boldsymbol{x}) + \varphi(\boldsymbol{x}))$ for any convex function $\varphi$, there are an infinite number of of DC decompositions for any DC function $f$. In general, the problem of finding a DC decomposition of a given function $f$ is difficult, however, for quadratic functions they can be easily found by using the eigenvalues of the quadratic coefficient matrix.

**Algorithm 1** Heuristic algorithm by López et al. [2]

**Step0:** Set $\delta := 0.05,\ \beta := 0.5,\ k := 1$.

**Step1:** Solve (CPP1) (time limit $10n$ sec.) and find a feasible solution $\boldsymbol{z}^{(0)} := (\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}, \boldsymbol{\alpha}^{(0)})$. Set $\boldsymbol{z}_{best} := \boldsymbol{z}^{(0)}$.

**Step2:** Solve (CPP1$_{\mathrm{R}}(\delta)$) (time limit $10n$ sec.) and find $\boldsymbol{z}^{(k)} := (\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, \boldsymbol{\alpha}^{(k)})$.

**Step3:** Round all values of $\alpha_i^{(k)}$ to 0 or 1 $(i = 1, \ldots, n)$.

**Step4:** If the rounded solution $\boldsymbol{z}^{(k)}$ is feasible for (CPP1), and $f(\boldsymbol{z}^{(k)}) < f(\boldsymbol{z}_{best})$, then update $\boldsymbol{z}_{best} := \boldsymbol{z}^{(k)}$.

**Step5:** If $\delta \leq 10^{-5}$ or $\boldsymbol{z}_{best}$ has not been updated for 3 successive iterations, output $\boldsymbol{z}_{best}$ and stop.

**Step6:** Set $k := k + 1,\ \delta := \beta\delta$ and goto **Step2**.

The class of DC functions is known to be very wide, for example, all twice continuously differentiable functions belong to the class of DC. Moreover, the DC property is preserved by many common function operations. The following theorem gives examples of DC-preserving operations, which will be subsequently used.

**Theorem 2.2** ([6])**.** For DC functions $f, f_i$ $(i = 1, \ldots, m)$, the functions defined by the following operations are also DC.

(i) $\displaystyle\sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{x}) \qquad (\lambda_i \in \mathbb{R}, i = 1, \ldots, m)$

(ii) $\displaystyle\max_{i=1,\ldots,m} f_i(\boldsymbol{x}), \quad \min_{i=1,\ldots,m} f_i(\boldsymbol{x})$

(iii) $|f(\boldsymbol{x})|, \quad f^+(\boldsymbol{x}) := \max\{0, f(\boldsymbol{x})\}, \quad f^-(\boldsymbol{x}) := \min\{0, f(\boldsymbol{x})\}$

We now consider optimization problems having a DC objective functions. The following problem is called a *DC program.*

$$\left|\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & f(\boldsymbol{x}) = g(\boldsymbol{x}) - h(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in S,\ \boldsymbol{x} \in \mathbb{R}^n \end{array}\right. \tag{2}$$

We assume that $g, h : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ are closed proper convex functions satisfying $\emptyset \neq \mathrm{dom}\, g \subseteq \mathrm{dom}\, h$, and that $S$ is a closed convex set. For technical reasons, we set $\infty - \infty = \infty$.

DC programs are an important class of nonconvex optimization problems, with a wide range of applications. They have been studied since the late 20th cetury (e.g. [6]), and are known to satisfy many nice mathematical properties, of which the most important is perhaps *Toland-Singer duality*. Tao et al. [6] have also proposed the *DCA*, which can efficiently find a local optimum.

In the following descriptions, for function $g$, we denote by $\partial g(\boldsymbol{x})$ the subdifferential of $g$ at $\boldsymbol{x}$, and by $g^* \text{ 或 } g$ the conjugate of $g$.

**Proposition 2.3** ([6])**.** For any optimal solution $\boldsymbol{x}^*$ of (2), the following properties hold.

1. $\partial g(\boldsymbol{x}^*) \supseteq \partial h(\boldsymbol{x}^*)$
2. $\bar{\boldsymbol{y}} \in \partial h(\boldsymbol{x}^*) \Leftrightarrow \boldsymbol{x}^* \in \partial h^*(\bar{\boldsymbol{y}})$
3. $\bar{\boldsymbol{y}} \in \partial h(\boldsymbol{x}^*) \Rightarrow \bar{\boldsymbol{y}}$ is an optimal solution of $\inf_{\boldsymbol{y}\in\mathbb{R}^n} \{h^*(\boldsymbol{y}) - g^*(\boldsymbol{y})\}$

**Theorem 2.4** (Toland-Singer duality, [6]))**.** The following properties hold for a DC function$f = g - h$.

$$\inf_{\boldsymbol{x}\in\mathbb{R}^n} \{g(\boldsymbol{x}) - h(\boldsymbol{x})\} = \inf_{\boldsymbol{y}\in\mathbb{R}^n} \{h^*(\boldsymbol{y}) - g^*(\boldsymbol{y})\} \tag{3}$$

**Definition 2.5.** For a DC function $f = g - h$, a *stationary DC point* is vector $\boldsymbol{x}^*$ satisfying the condition

$$\partial g(\boldsymbol{x}^*) \cap \partial h(\boldsymbol{x}^*) \neq \emptyset. \tag{4}$$

The DCA, due to Tao et al. [6], which finds a stationary point of the problem (2) is described in Algorithm 2.

**Algorithm 2** DCA

**Step0:** Choose $\boldsymbol{x}^{(0)}$, and set $k := 0$.

**Step1:** Choose $\boldsymbol{y}^{(k)} \in \partial h(\boldsymbol{x}^{(k)})$.

**Step2:** Find $\boldsymbol{x}^{(k+1)} \in \partial g^*(\boldsymbol{y}^{(k)})$.

**Step3:** If $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\| < \epsilon$, then output $\boldsymbol{x}^{(k+1)}$ and stop.
Otherwise set $k := k + 1$ and goto **Step1**.

Upon termination, this algorithm finds an $\boldsymbol{x}^*$ satisfying (4). In the case that both $g$ and $h$ are smooth, equation (4) is equivalent to the relation $\nabla g(\boldsymbol{x}^*) = \nabla h(\boldsymbol{x}^*)$, thus the solution $\boldsymbol{x}^*$ output by the DCA will satisfy $\nabla f(\boldsymbol{x}^*) = \nabla g(\boldsymbol{x}^*) - \nabla h(\boldsymbol{x}^*) = 0$.

We now consider *discrete* DC programs, that is, DC programs in which all variables are restricted to integer values. A discrete DC program can be formulated as follows:

$$\left| \begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & f(\boldsymbol{x}) = g(\boldsymbol{x}) - h(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in S,\ \boldsymbol{x} \in \mathbb{Z}^n \end{array} \right. . \tag{5}$$

Again, we assume that $g, h : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ are closed proper convex functions with $\emptyset \neq \operatorname{dom} g \subseteq \operatorname{dom} h$, and that $S$ is a closed convex set. Solving this problem usually involves the use of some type of continuous relaxation. The obvious relaxation which replaces $\boldsymbol{x} \in \mathbb{Z}^n$ with $\boldsymbol{x} \in \mathbb{R}^n$, does not work well in general, since an integrality gap often occurs. However, this is not the case for the relaxation based on the closed convex hull extension.

**Definition 2.6.** Let $g$ be a discrete function $g : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$. If there exists a (continous) closed convex function $\hat{g} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ satisfying the condition

$$\hat{g}(\boldsymbol{x}) = g(\boldsymbol{x}) \quad (\boldsymbol{x} \in \mathbb{Z}^n), \tag{6}$$

then $g$ is said to be *convex extensible*, and $\hat{g}$ is called a *closed convex extension* of $g$.

In general, a convex extensible discrete function has many convex extensions. Among these, the following closed convex hull extension is of particular importance.

**Definition 2.7.** For a convex extensible discrete function $g : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$, the *closed convex hull extension* is the continuous closed convex function $g^{\mathrm{cl}} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ satisfying

$$g^{\mathrm{cl}}(\boldsymbol{x}) = \sup\bigl\{\alpha + \langle \boldsymbol{p}, \boldsymbol{x} \rangle \,|\, \alpha + \langle \boldsymbol{p}, \boldsymbol{y} \rangle \leq g(\boldsymbol{y}) \quad (\boldsymbol{y} \in \mathbb{Z}^n)\bigr\}. \tag{7}$$

The next theorem shown by Maehara et al. [4], shows that continuous relaxations using closed convex hull extensions have no integrality gap.

**Theorem 2.8** ([4])**.** Let $g, h : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ be convex extensible discrete functions such that $\operatorname{dom} g$ is bounded, and $\operatorname{dom} g \subseteq \operatorname{dom} h$. Then,

$$\inf_{\boldsymbol{x} \in \mathbb{Z}^n} \bigl\{g(\boldsymbol{x}) - h(\boldsymbol{x})\bigr\} = \inf_{\boldsymbol{x} \in \mathbb{R}^n} \bigl\{g^{\mathrm{cl}}(\boldsymbol{x}) - \hat{h}(\boldsymbol{x})\bigr\} \tag{8}$$

for any closed convex extension $\hat{h}$ of $h$. Thus, we can solve (5) by solving a continous relaxation in which $g$ is replaced by its closed convex hull extension (and $h$ by any closed convex extension).

In [4], Maehara et al. proposed an algorithm for discrete DC programs, based on the above theorem and DCA of [6].

Finally, we introduce the algorithm of Okuno et. al for mixed integer DC programs. A mixed integer DC program is a DC program having both discrete and continous variables:

$$\left|\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & f(\boldsymbol{x}) = g(\boldsymbol{x}) - h(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} = (\boldsymbol{x}_M, \boldsymbol{x}_N) \in S \\ & \boldsymbol{x}_M \in \mathbb{R}^M, \boldsymbol{x}_N \in \mathbb{Z}^N \end{array}\right. \tag{9}$$

Here, $n = |M| + |N|$, functions $g, h : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ are closed convex functions with $\emptyset \neq \text{dom } g \subseteq \text{dom } h$, and $S \subseteq \mathbb{R}^n$ is a closed convex set.

In [5] Okuno et al. extended the results of Maehara et al.[4] by proving that Theorem 2.8 can be extended to the above problem, proposing the following MIDCA.

**Algorithm 3** MIDCA

**Step0:** Choose $\boldsymbol{x}^{(0)}$, and set $k := 0$.

**Step1:** Choose $\boldsymbol{y}^{(k)} \in \partial h(\boldsymbol{x}^{(k)})$.

**Step2:** Find a solution $\boldsymbol{x}^{(k+1)}$ of the following subproblem:

$$\left|\begin{array}{l} \underset{\boldsymbol{x}}{\text{minimize}} \;\; g(\boldsymbol{x}) - \left(h(\boldsymbol{x}^{(k)}) + \boldsymbol{y}^{(k)\top}(\boldsymbol{x} - \boldsymbol{x}^{(k)})\right) \\ \text{subject to } \boldsymbol{x} = (\boldsymbol{x}_M, \boldsymbol{x}_N) \in S, \boldsymbol{x}_M \in \mathbb{R}^M, \boldsymbol{x}_N \in \mathbb{Z}^N \end{array}\right.$$

**Step3:** If $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\| < \epsilon$, then output $\boldsymbol{x}^{(k+1)}$ and stop.
Otherwise, set $k := k + 1$ and go to **Step1**.

In each iteration, we find a solution $\boldsymbol{x}^{(k+1)}$, of the mixed integer convex problem approximating (9) at $\boldsymbol{x}^{(k)}$. Thus, each $\boldsymbol{x}^{(k)}$ is always feasible to (9) . Upon termination, the algorithm outputs a MIDC stationary point satisfying (4).

# 3 New heuristic algorithms for the circle packing problem

In this section, we propose two heuristic algorithms for the circle packing problem.

In doing so, we first reformulate (CPP1) as a mixed integer DC program, then modify some varables and constraints. The first algorithm alternately solves two problems; one is the modified MIDC reformulation of (CPP1), which we solve by the MIDCA, and the other is a related problem, in which we pack *all* circles into the container so as to minimize the overlapping area. The idea is to find a good feasible solution of (CPP1), then move the rejected circles into positions which can be used to improve it. The second algorithm has the same framework, but in solving the MIDC reformulation, we replace the MIDCA by a method which alternately solves a series of linear integer programs, and continuous DC programs.

We begin by describing the MIDC reformulation of (CPP1).

## 3.1 Refomulation as an MIDC program

As mentioned in Section 2, the formulation (CPP1) is a nonconvex quadratic program, which is very difficult to solve. Much of the difficulty stems from the nonconvex quadratic constraint (F2). We eliminate this complication by linearizing it through means of an auxiliary variable $\beta ij$. More explicitly, we replace the quadratic term $\alpha_i\alpha_j$ by $\beta_{ij}$, and add some constraints that force $\beta_{ij} = \alpha_i\alpha_j$, as below.

$$\begin{cases} \beta_{ij}(R_i + R_j)^2 \le (x_i - x_j)^2 + (y_i - y_j)^2 & (1 \le i < j \le n) \quad \text{(F2a)} \\ \alpha_i + \alpha_j - 1 \le \beta_{ij} & (1 \le i < j \le n) \quad \text{(F2b)} \\ 0 \le \beta_{ij} \le \alpha_i & (1 \le i < j \le n) \quad \text{(F2c)} \\ 0 \le \beta_{ij} \le \alpha_j & (1 \le i < j \le n) \quad \text{(F2d)} \end{cases}$$

If we replace the constrant (F2) by the inequalities (F2a) – (F2d), we have got rid of the quadratic terms, but not the nonconvexity. We deal with this, by penalizing the amount of violation of constraint (F2a), and incorporating it into the objective function:

$$f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = -\pi \sum_{i=1}^{n} \alpha_i R_i^2 + \sum_{1 \le i < j \le n} P_{ij} \max\{0, \beta_{ij}(R_i + R_j)^2 - (x_i - x_j)^2 - (y_i - y_j)^2\}$$

Here, $P_{ij}$ denotes the penalty parameter. Let (CPP2) be the problem obtained by replacing in (CPP1), the constraint (F2) by (F2b) – (F2d), and the objective function (F1) by the above function. We now express the objective function as a DC function.

$$f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = -\pi \sum_{i=1}^{n} \alpha_i R_i^2 + \sum_{1 \le i < j \le n} P_{ij} \max\{(x_i - x_j)^2 + (y_i - y_j)^2, \beta_{ij}(R_i + R_j)^2\} - \sum_{1 \le i < j \le n} P_{ij}\{(x_i - x_j)^2 + (y_i - y_j)^2\}$$

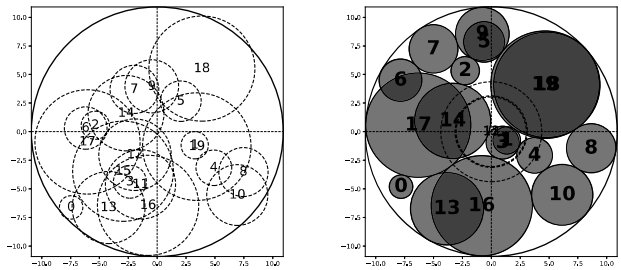

Figure 1: Solution of ($\mathrm{CPP2_{DC}}$): lnitial solution ⇒ MIDC stationary point

This function contains the max operator, and hence is not smooth. We add another auxiliary variable $u_{ij}$ to deal with this. The resulting problem can be described as below.

$$
\begin{array}{ll}
\underset{\boldsymbol{x},\boldsymbol{y},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{u}}{\text{minimize}} & \\
\quad -\pi\sum_{i=1}^{n}\alpha_i R_i^2 + \sum_{1\le i<j\le n} P_{ij}u_{ij} - \sum_{1\le i<j\le n} P_{ij}\{(x_i-x_j)^2+(y_i-y_j)^2\} & \text{(F7)} \\
\text{subject to} \quad (x_i-x_j)^2+(y_i-y_j)^2 \le u_{ij} \qquad (1\le i<j\le n) & \text{(F2e)} \\
\quad \beta_{ij}(R_i+R_j)^2 \le u_{ij} \qquad (1\le i<j\le n) & \text{(F2f)} \\
\quad \text{constraints (F3),(F4),(F5),(F2b),(F2c),(F2d),(F6)} &
\end{array}
$$

We will call this problem ($\mathrm{CPP2_{DC}}$). Unfortunately, directly applying the MIDCA to this problem did not lead to good results. An example with $n = 20$ circles is given in Figure 1 to illustrate this. In our implementation, we chose all $(x_i, y_i)$ as random values in the segment $[-(R_0 - R_i), R_0 - R_i]$. The figure on the left side shows the initial solution, and the right side the MIDC stationary point. In the MIDC stationary point, all circles not centered at the origin have $\alpha_i = 1$, and are chosen to be placed. However, there is a great deal of overlap, which no amount of adjusting the penalty parameters $P_{ij}$ was able to resolve. The reason behind this phenomenon is that constraints (F3),(F4),(F5) force all rejected circles to be centered at the origin, which means that all circles *not* centered at the origin are necessarily chosen to be placed, regardless of the amount of overlap. Moreover, the MIDCA seems to be unable to change variable values by large amounts, thus circles placed far from the origin cannot move there.

Since there is no imperative reason for rejected circles to be situated at the origin, we alter the constraints (F3),(F4),(F5) as below, and allow unplaced circles

to be placed anywhere within the container.

$$\begin{cases} x_i^2 + y_i^2 \le (R_0 - R_i)^2 & (i = 1, 2, \ldots, n) \quad \text{(F3a)} \\ -(R_0 - R_i) \le x_i \le R_0 - R_i & (i = 1, 2, \ldots, n) \quad \text{(F4a)} \\ -(R_0 - R_i) \le y_i \le R_0 - R_i & (i = 1, 2, \ldots, n) \quad \text{(F5a)} \end{cases}$$

The resulting problem which we will call (CPP3$_{\mathrm{DC}}$), can be explicitly written as below

$$\begin{aligned} &\underset{\boldsymbol{x},\boldsymbol{y},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{u}}{\text{minimize}} \\ &\quad -\pi \sum_{i=1}^{n} \alpha_i R_i^2 + \sum_{1 \le i < j \le n} P_{ij} u_{ij} - \sum_{1 \le i < j \le n} P_{ij} \{(x_i - x_j)^2 + (y_i - y_j)^2\} && \text{(F7)} \\ &\text{subject to} \quad (x_i - x_j)^2 + (y_i - y_j)^2 \le u_{ij} \quad (1 \le i < j \le n) && \text{(F2e)} \\ &\qquad \beta_{ij}(R_i + R_j)^2 \le u_{ij} \quad (1 \le i < j \le n) && \text{(F2f)} \\ &\qquad x_i^2 + y_i^2 \le (R_0 - R_i)^2 \quad (i = 1, 2, \ldots, n) && \text{(F3a)} \\ &\qquad -(R_0 - R_i) \le x_i \le R_0 - R_i \quad (i = 1, 2, \ldots, n) && \text{(F4a)} \\ &\qquad -(R_0 - R_i) \le y_i \le R_0 - R_i \quad (i = 1, 2, \ldots, n) && \text{(F5a)} \\ &\qquad \alpha_i + \alpha_j - 1 \le \beta_{ij} \quad (1 \le i < j \le n) && \text{(F2b)} \\ &\qquad 0 \le \beta_{ij} \le \alpha_i \quad (1 \le i < j \le n) && \text{(F2c)} \\ &\qquad 0 \le \beta_{ij} \le \alpha_j \quad (1 \le i < j \le n) && \text{(F2d)} \\ &\qquad \alpha_i \in \{0, 1\} \quad (i = 1, 2, \ldots, n) && \text{(F6)} \end{aligned}$$

In Figure 2, we show the results of applying the MIDCA to (CPP3$_{\mathrm{DC}}$), with the same problem data and initial solution of Figure 1. The resulting solution is now feasible, but far from good. The reason is that the stationary point found by the MIDCA depends a great deal on the initial solution. We next introduce the minimum overlap problem, and show how it can be used to obtain a good initial solution.

## 3.2 Obtaining good initial solutions: the minimum overlap problem

In observing the results of the MIDCA, we notice that for a given initial solution, it seems mainly to make minor place adjustments, then choose the circles which do not overlap each other, to be placed. Thus, by using configurations with small overlap as initial solutions, we may expect to obtain good MIDC stationary points.

The next problem, which we call (OVERLAP1), attempts to find the configuration in which *all* circles are placed within the container, with a minimum

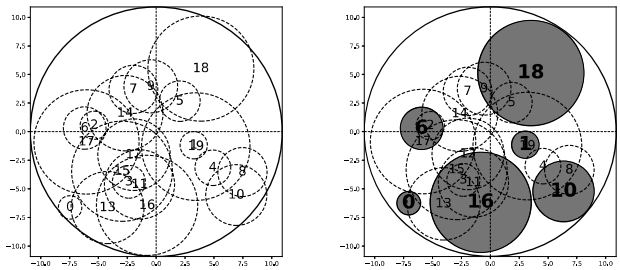

Figure 2: Solution of ($\mathrm{CPP3_{DC}}$): lnitial solution $\Rightarrow$ MIDC stationary point

amount of overlap. Thus, it is a nonlinear program having only continuous variables $(x_i, y_i)$ $(i = 1, \ldots, n)$ corresponding to the center coordinates of the circles.

$$\left|\begin{array}{lll} \underset{\boldsymbol{x},\boldsymbol{y}}{\text{minimize}} & \displaystyle\sum_{1\le i<j\le n} \max\{0, (R_i+R_j)^2 - (x_i-x_j)^2 - (y_i-y_j)^2\} & \\ \text{subject to} & x_i^2 + y_i^2 \le (R_0 - R_i)^2 & (i = 1, 2, \ldots, n) \\ & -(R_0 - R_i) \le x_i \le R_0 - R_i & (i = 1, 2, \ldots, n) \\ & -(R_0 - R_i) \le y_i \le R_0 - R_i & (i = 1, 2, \ldots, n) \end{array}\right.$$

The objective function of the above problem is both nonsmooth and nonconvex. By using the same techniques as in Section 3.1, we obtain the following DC program, which we call ($\mathrm{OVERLAP1_{DC}}$).

$$\left|\begin{array}{lll} \underset{\boldsymbol{x},\boldsymbol{y},\boldsymbol{u}}{\text{minimize}} & \displaystyle\sum_{1\le i<j\le n} u_{ij} - \sum_{1\le i<j\le n} \{(x_i-x_j)^2 + (y_i-y_j)^2\} & \\ \text{subject to} & (x_i-x_j)^2 + (y_i-y_j)^2 \le u_{ij} & (1 \le i < j \le n) \\ & (R_i+R_j)^2 \le u_{ij} & (1 \le i < j \le n) \\ & x_i^2 + y_i^2 \le (R_0 - R_i)^2 & (i = 1, 2, \ldots, n) \\ & -(R_0 - R_i) \le x_i \le R_0 - R_i & (i = 1, 2, \ldots, n) \\ & -(R_0 - R_i) \le y_i \le R_0 - R_i & (i = 1, 2, \ldots, n) \end{array}\right.$$

As ($\mathrm{OVERLAP1_{DC}}$) has only continuous variables, it can be solved by the DCA using any nonlinear optimization solver. In Figure 3, we show the result of solving ($\mathrm{OVERLAP1_{DC}}$) using the same problem data and initial solution of Figures 1 and 2. By then using the DC stationary point as the initial solution of ($\mathrm{CPP3_{DC}}$), the MIDC stationary point shown in Figure 4 was obtained. This is clearly a much better solution than that of Figure 2. However, there is still room for improvement.

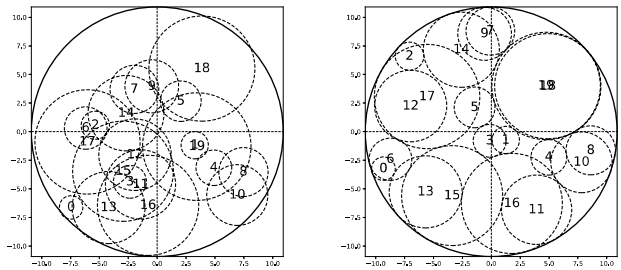

Figure 3: Solution of (OVERLAP1): Initial solution ⇒ DC stationary point

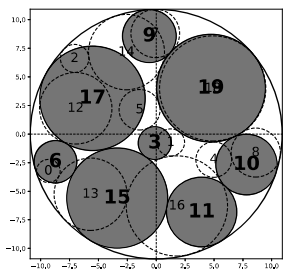

Figure 4: Solution of ($\mathrm{CPP3_{DC}}$)(Initial solution: DC stationary point of Fig. 3)

In the next section, we describe our first algorithm, which attempts to obtain improved solutions by alternately 'moving' the rejected circles so as to minimize overlap with the chosen circles, then re-solving problem ($\mathrm{CPP3_{DC}}$).

## 3.3 Our first method

In the previous subsection, we have shown how to obtain a solution by using the DC stationary point of the minimum overlap problem ($\mathrm{OVERLAP1_{DC}}$) as an initial solution of ($\mathrm{CPP3_{DC}}$). We now consider how to further improve this solution.

In order to improve the present solution, we use the minimum overlap problem in which the placement of chosen circles is fixed.

Let $\bar{\boldsymbol{z}} = (\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{\alpha}}, \bar{\boldsymbol{\beta}}, \bar{\boldsymbol{u}})$ be the present solution, and denote by $S$ and $T$ respectively the sets of unplaced and placed circles in $\bar{\boldsymbol{z}}$, that is, $S = \{i \mid \bar{\alpha}_i = 0\}$ and $T = \{j \mid \bar{\alpha}_j = 1\}$. We fix the circles in $T$, and solve the problem minimizing overlap between placed and unplaced circles. More precisely, we fix the values $(x_j, y_j) = (\bar{x}_j, \bar{y}_j)$ $(j \in T)$, and solve the problem having variables $(x_i, y_i)$ $(i \in S)$,

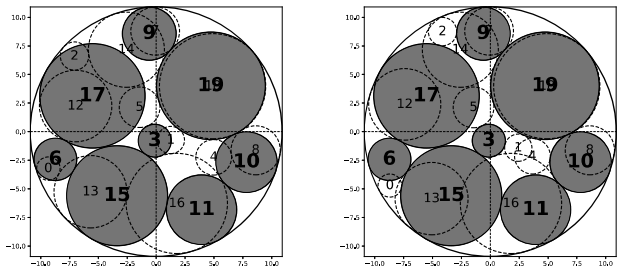

Figure 5: Solution of (OVERLAP2) : Initial (Fig. 2) ⇒ DC stationary point

objective function

$$\sum_{i\in S}\sum_{j\in T}\max\{0,(R_i+R_j)^2-(x_i-\bar{x_j})^2-(y_i-\bar{y_j}^2\},$$

and the same constraints as (OVERLAP1). Let us call this problem (OVERLAP2($\bar{z}$)). The objective of moving the unplaced circles so as to minimize the overlapping area with the fixed circles, is based on the idea that some of the previously rejected circles will move to positions having little overlap with the placed circles, allowing us to add them to the set of chosen circles. As with (OVERLAP1), we solve (OVERLAP2($\bar{z}$)) by reformulating it as a DC program and using a nonlinear optimization solver. In Figure 5, we show the results of solving (OVERLAP2) arising from the MIDC stationary point shown in Figure 2. We may observe the unplaced circles, indicated by dotted lines, moving so as to reduce overlap with the placed circles. Using the resulting configuration as the initial solution to problem ($\text{CPP3}_{\text{DC}}$) and again applying the MIDC algorithm, we obtain the MIDC stationary point of Figure 6. This solution is an improvement on Figure 2, in that circles marked 2 and 4 are newly chosen.

Basically, our first algorithm iterates this procedure until no further improvement occurs. The total framework can be formally described as below. Here, $f_{\text{DC}}(\boldsymbol{z})$ denotes the objective function (F7) of problem ($\mathbf{CPP3_{DC}}$).

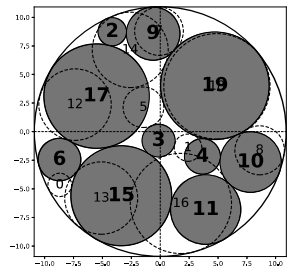


Figure 6: MIDC point of (CPP3$_{\mathrm{DC}}$): (Initial solution: Fig 5)

**Algorithm 4** Proposed Method 1

**Step0:** Choose an arbitrary initial solution $\boldsymbol{z}^{(0)} := (\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}, \boldsymbol{\alpha}^{(0)}, \boldsymbol{\beta}^{(0)}, \boldsymbol{u}^{(0)})$, and set $k := 0$.

**Step1:** Solve (OVERLAP1$_{\mathrm{DC}}$), using $\boldsymbol{z}^{(0)}$ as an initial solution, and find a DC stationary point $\boldsymbol{z}' := (\boldsymbol{x}', \boldsymbol{y}', \boldsymbol{\alpha}^{(0)}, \boldsymbol{\beta}^{(0)}, \boldsymbol{u}')$ with small overlap. Set $\boldsymbol{z}^{(0)} = \boldsymbol{z}'$.

**Step2:** Apply the MIDC algorithm to (CPP3$_{\mathrm{DC}}$), and find an MIDC stationary point $\boldsymbol{z}^{(k+1)} := (\boldsymbol{x}^{(k+1)}, \boldsymbol{y}^{(k+1)}, \boldsymbol{\alpha}^{(k+1)}, \boldsymbol{\beta}^{(k+1)}, \boldsymbol{u}^{(k+1)})$.

**Step3:** If $f_{\mathrm{DC}}(\boldsymbol{z}^{(k)}) = f_{\mathrm{DC}}(\boldsymbol{z}^{(k+1)})$, then output $\boldsymbol{z}^{(k+1)}$ and stop.

**Step4:** Solve the problem (OVERLAP2($\boldsymbol{z}^{(k+1)}$) to find a DC stationary point $\boldsymbol{z}' := (\boldsymbol{x}', \boldsymbol{y}', \boldsymbol{\alpha}^{(k+1)}, \boldsymbol{\beta}^{(k+1)}, \boldsymbol{u}')$, set $\boldsymbol{z}^{(k+1)} := \boldsymbol{z}'$, $k := k + 1$ and goto **Step2**.

In the instance shown in the previous figures, the algorithm runs through two iterations of Steps 2 and 3, to output the solution shown in Figure 7.

## 3.4 Our second method

In this section, we propose a method with the same framework as our first method, but insetad of solving problem (CPP3$_{\mathrm{DC}}$), we introduce a procedure which alternately solves a series of *linear* mixed integer problems and *continuous* DC problems.

We begin by noting that in the DC decomposition of the objective function of

(CPP3$_{\rm DC}$), the two convex functions can be seen as having disjoint sets of variables, that is,

$$f(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{u}) = g(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{u}) - h(\boldsymbol{x},\boldsymbol{y}),$$
$$g(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{u}) = -\pi\sum_{i=1}^{n}\alpha_i R_i^2 + \sum_{1\le i<j\le n} P_{ij}u_{ij},$$
$$h(\boldsymbol{x},\boldsymbol{y}) = \sum_{1\le i<j\le n} P_{ij}\{(x_i-x_j)^2+(y_i-y_j)^2\}$$

Moreover, the function $g(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{u})$ has integer variables, but is linear, and the quadratic function $h(\boldsymbol{x},\boldsymbol{y})$ contains only continous variables. In our algorithm, we take advantage of this fact to speedup Step 2 (solving (CPP3$_{\rm DC}$)) of our first algorithm.

Instead of using the MIDCA, we employ the following iterative two-step procedure to solve (CPP3$_{\rm DC}$)). First, we solve the linear mixed integer program obtained by fixing the values of $(\boldsymbol{x},\boldsymbol{y})$ in (CPP3$_{\rm DC}$). This problem, which we call (CPP4), has variables$(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{u})$, and can be solved by any MIP solver. Next, we fix all values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ in (CPP3$_{\rm DC}$) and solve the resulting problem, a continuous DC program which we call (OVERLAP3$_{\rm DC}$). This can be accomplished by any nonlinear optimization solver. We then iterate the above procedure until there is no more change in the solutions.

Step 2 of our second algorithm can be formally described as follows. The remaining steps (0, 1, 3, and 4) are identical to our first method.

**Algorithm 5 Step 2** of our second method

**Step 2:** ($\boldsymbol{z}^{(k)} = (\boldsymbol{x}^{(k)},\boldsymbol{y}^{(k)},\boldsymbol{\alpha}^{(k)},\boldsymbol{\beta}^{(k)},\boldsymbol{u}^{(k)})$ is the present solution.)
Set $j := 0$, $\bar{\boldsymbol{z}}^{(0)} = (\bar{\boldsymbol{x}}^{(0)},\bar{\boldsymbol{y}}^{(0)},\bar{\boldsymbol{\alpha}}^{(0)},\bar{\boldsymbol{\beta}}^{(0)},\bar{\boldsymbol{u}}^{(0)}) := \boldsymbol{z}^{(k)}$, and find $\boldsymbol{z}^{(k+1)}$ by the following procedure.

**Step2.1:** Fix $(\boldsymbol{x},\boldsymbol{y}) = (\bar{\boldsymbol{x}}^{(j)},\bar{\boldsymbol{y}}^{(j)})$ in (CPP3$_{\rm DC}$), and use a MIP solver to find $(\bar{\boldsymbol{\alpha}}^{(j+1)},\bar{\boldsymbol{\beta}}^{(j+1)},\bar{\boldsymbol{u}}^{(j+1)})$.

**Step2.2:** Fix $(\boldsymbol{\alpha},\boldsymbol{\beta}) = (\bar{\boldsymbol{\alpha}}^{(j+1)},\bar{\boldsymbol{\beta}}^{(j+1)})$ in (CPP3$_{\rm DC}$),
and use a nonlinear solver to find $(\bar{\boldsymbol{x}}^{(j+1)},\bar{\boldsymbol{y}}^{(j+1)},\bar{\boldsymbol{u}}^{(j+1)})$.
Set $\bar{\boldsymbol{z}}^{(j+1)} := (\bar{\boldsymbol{x}}^{(j+1)},\bar{\boldsymbol{y}}^{(j+1)},\bar{\boldsymbol{\alpha}}^{(j+1)},\bar{\boldsymbol{\beta}}^{(j+1)},\bar{\boldsymbol{u}}^{(j+1)})$.

**Step2.3:** If $f_{\rm DC}(\bar{\boldsymbol{z}}^{(j)}) = f_{\rm DC}(\bar{z}^{(j+1)})$, then set $\boldsymbol{z}^{(k+1)} := \bar{\boldsymbol{z}}^{(j+1)}$ and goto **Step3**, otherwise set $j := j+1$ and goto **Step2.1**.

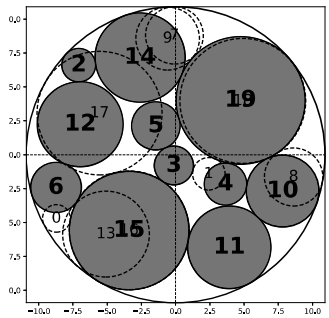


Figure 7: Final solution

# 4 Computational experiments

In this section, we report the results of numerical experiments. The computer we used is a Windows10, 4.00 gigahertz PC with Intel i7-6700K core and 16 gigabyte memory. All implmentations were coded with Python 3.6, and the solver we used is Gurobi7.5.2 (both MIP and nonlinear optimization). For our problem data, we used the circle packing problems originally used in [2], and available from the OR-Library[7].

Our experiments consist of two sets; in the first we compared our new two new algorithms with with the existing method of López et al. [2]. Results showed that our second method showed most promise, thus, in the next set, we conducted more experiments on this algorithm in depth.

## 4.1 The first set of experiments: comparison of algorithms

In order to investigate the efficiency and solution quality of our two new algorithms, we compared them with the algorithm of López et. al [2] as well as direct application of SCIP, and the MIDCA. The test problems we used are the same as those used in [2]. They involve $n = 10, 20, 30$ circles, and are available from the OR-Library[7]. Each test problem consists of a set of $n$ circles and three values for the container radius $R_0$, thus there are a total of nine data sets, which are specified as $n-1, 2, 3$. For each of these data sets, we randomly generated 5 initial solutions, and evaluated the performance by the average CPU time(seconds), and the objective function value of the best solution found. A limit of 3600 seconds was set on all CPU time. Table 1 summarizes the results. For some reason, which we could not determine, the solutions found by our implementation of the López et. al algorithm were distinctly inferior to the results given in the paper [2]. Thus, in the interest of fairness, for this algorithm we have reproduced the best value re-

Table 1: Results for $n = 10, 20, 30$ circles(5 initial solutions)

| data | results | SCIP | López | MIDCA | Method 1 | Method 2 |
|---|---|---|---|---|---|---|
| 10-1 | avg. time | 12.440 | – | 0.919 | 2.945 | 1.606 |
| | best value | 83.080 | 83.080 | 77.455 | 83.080 | 83.080 |
| 10-2 | avg. time | 15.650 | – | 0.840 | 4.803 | 2.297 |
| | best value | 124.620 | 124.620 | 110.140 | 120.956 | 120.956 |
| 10-3 | avg. time | 2596.380 | – | 0.586 | 2.423 | 0.660 |
| | best value | 197.071 | 197.071 | 138.590 | 193.757 | 193.757 |
| 20-1 | avg. time | 3600 | – | 20.569 | 162.322 | 6.297 |
| | best value | 105.480 | 143.755 | 101.184 | 141.139 | 141.139 |
| 20-2 | avg. time | 3600 | – | 23.754 | 76.034 | 5.960 |
| | best value | 117.678 | 219.065 | 176.508 | 215.571 | 218.086 |
| 20-3 | avg. time | 3600 | – | 27.064 | 61.940 | 10.661 |
| | best value | 211.069 | 290.506 | 207.758 | 295.617 | 295.617 |
| 30-1 | avg. time | 3600 | – | 338.404 | 1437.161 | 21.498 |
| | best value | 210.236 | 254.244 | 184.695 | 250.774 | 250.774 |
| 30-2 | avg. time | 3600 | – | 680.300 | 2071.709 | 28.850 |
| | best value | 232.727 | 372.064 | 261.090 | 376.204 | 377.217 |
| 30-3 | avg. time | 3600 | – | 551.343 | 1824.820 | 51.066 |
| | best value | 261.206 | 502.017 | 324.590 | 493.377 | 493.377 |

sults from [2] (CPU times are omitted, as compuating environments are different). In Table 1, abbreviations are as follows.

**SCIP:** results of directly solving (CPP1)with SCIP5.0.0,
**López:** results (best value only) as reported in [2], of solving (CPP1) with the algorithm of López et. al,
**MIDC:** results of solving ($\text{CPP3}_{\text{DC}}$) by the MIDCA,
**Method 1:** results of solving ($\text{CPP3}_{\text{DC}}$) by our first method,
**Method 2:** results of solving ($\text{CPP3}_{\text{DC}}$) by our second method.

Obviously, simply using the MIDCA on ($\text{CPP3}_{\text{DC}}$) does not work, as the solution values are much worse than the three heuristic methods, also, finding exact solutions for $n \geq 20$ circles by SCIP is also prohibitively time-consuming. In comparing the heuristic methods by the best value found, we see that the method of [2] is slightly better than ours, but not by much. As to our two methods, the second heuristic is clearly superior, in both that it is much faster, and the solutions quality is at least as good as the first, and sometimes better.

Thus, we focused on our second method, and conducted more experiments to determine the practical limit of the value of $n$ for which it can be used.

Table 2: Results for $n = 30, 40$ circles (50 initial solutions)

| data | results | López | Method 2 | data | results | López | Method 2 |
|---|---|---|---|---|---|---|---|
| 30-1 | avg. time | – | 21.648 | 40-1 | avg. time | – | 83.814 |
| | best value | 254.244 | 254.477 | | best value | 318.914 | 340.589 |
| 30-2 | avg. time | – | 32.344 | 40-2 | avg. time | – | 137.065 |
| | best value | 372.064 | 383.863 | | best value | 486.941 | 507.734 |
| 30-3 | avg. time | – | 47.031 | 40-3 | avg. time | – | 202.583 |
| | best value | 502.017 | 512.127 | | best value | 642.909 | 680.293 |

Table 3: Results for $n = 50, 60$ circles (10 initial solutions)

| data | results | Method 2 | data | results | Method 2 |
|---|---|---|---|---|---|
| 50-1 | avg. time | 277.009 | 60-1 | avg. time | 817.901 |
| | best value | 444.540 | | best value | 495.121 |
| 50-2 | avg. time | 592.689 | 60-2 | avg. time | 1635.427 |
| | best value | 649.954 | | best value | 726.926 |
| 50-3 | avg. time) | 820.441 | 60-3 | avg. time | 2286.504 |
| | best value | 870.329 | | best value | 984.102 |

## 4.2 The second set of experiments: testing on data with large $n$

We give results of experiments on data with large $n$.

First, in Table 2, we show results for data with $n = 30, 40$ circles, comparing our second algorithm and the López et. al method. For our algorithm, we randomly generated 50 initial solutions, and give the average CPU time (seconds) and best solution value found. Best values for the López et. al algorithm are again, those reported in [2]. We notice that for all data, our method was able to find better solutions, within reasonable computation time.

Next, to determine the practial limit of the solvable data size, we applied our algorithm to problems with $n = 50, 60$ circles. The data was generated by the same principle used in [2] to generate the previous twelve sets: radii for the $n$ circles were randomly generated to two decimal places from $[1, 5]$, and the container radius $R_0$ was set to values so that the area of the container is approximately equal to $1/3, 1/2$ and $2/3$ of the total area $\sum_{i=1}^{n} \pi R_i^2$ of the $n$ circles. For each data set, we generated 10 random initial solutions. Average CPU times (in seconds) and the objective values of best solutions are given in Table 3. As no results for these values of $n$ are given in [2], we show results only for our second method. While there is no accurate way to evaluate the quality of the solutions, drawings of the actual configurations suggest that they are reasonably good. On the other hand, the increase in the computational time between $n = 50$ and $n = 60$ is very large,

which leads us to conclude that $n = 60$ is near the limit for which our algorithm can be practically applied.

## 5 Concluding remarks

In this paper, we have considered the problem of packing a set of unequal circles into a circular container, and shown how it can be formulated as a mixed integer DC program. Although simple application of the MIDCA did not produce good results, we further introduced the minimum overlap problem, proposed a heurstic algorithm which alternately solves the two problems and demonstrated through computational experiments that it find good solutions. We then further improved this algorithm by replacing the time-consuming MIDCA with a procedure which alternately solves linear mixed integer programs and continuous DC programs. Experiments confirm that the improved method is applicable to instances with up to $n = 60$ circles.

Further measures to speed up the algorithm, as well as methods to make the algorithm more robust are possible subjects for future exploration.

## References

[1] A. Lodi, S. Martello, M. Monaci: Two-dimensional packing problems: A survey. European journal of operational research, **141-2** (2002), 241-252.

[2] C.O. López, J.E. Beasley: A formulation space search heuristic for packing unequal circles in a fixed size circular container. European Journal of Operational Research, **251-1** (2016), 64–73.

[3] H. Dyckhoff: A typology of cutting and packing problems. European Journal of Operational Research, **44-2** (1990), 145-159.

[4] T. Maehara, N. Marumo, K. Murota: Continuous relaxation for discrete DC programming. Modelling, Computation and Optimization in Information Systems and Management Sciences, (2015), 181–190.

[5] T. Okuno, Y.T. Ikebe: A new approach for solving mixed integer DC programs using a continuous relaxation with no integrality gap and smoothing techniques. arXiv preprint arXiv:1702.00553, (2017).

[6] P.D. Tao, L.T.H. An: Convex analysis approach to dc programming. Acta Mathematica Vietnamica, **22-1** (1997), 289–355.

[7] OR-Library, http://people.brunel.ac.uk/˜mastjjb/jeb/orlib/circleinfo.html, (2018/01/24)