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

Satoru Masuda[a], Yoshiko Ikebe[a,*], Takayuki Okuno[b]

[a] *Tokyo University of Science, 6-3-1 Niijuku, Katsushika-ku, Tokyo 125-8585, Japan*
[b] *RIKEN, Center for Advanced Intelligence Project, 1-4-1 Nihonbashi, Chuo-ku, Tokyo 103-0027, Japan*

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. Here we propose heuristic algorithms 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.

Keywords: Circle packing, Nonlinear optimization, Mixed integer DC programming

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 [3, 2]. 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. [4]. In this problem, we have a set of n circles with radii

*Corresponding author
Email address: yoshiko@rs.tus.ac.jp (Yoshiko Ikebe)

$R_1, R_2, \ldots, R_n$, 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. [4] 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 for this problem 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. [6]. By applying our proposed 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. [4], 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 [4], 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 placed.

By using these variables, the problem, which we call (CPP1), can be formulated as

follows.

$$
\text{(CPP1)}\quad
\left|\begin{array}{lll}
\underset{\boldsymbol{x},\boldsymbol{y},\boldsymbol{\alpha}}{\text{minimize}} & f(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\alpha}) = -\pi\displaystyle\sum_{i=1}^{n}\alpha_i R_i^2 & \text{(F1)}\\
\text{subject to} & \alpha_i\alpha_j(R_i+R_j)^2 \le (x_i-x_j)^2+(y_i-y_j)^2 \quad (1\le i<j\le n) & \text{(F2)}\\
& x_i^2+y_i^2 \le \alpha_i(R_0-R_i)^2 \quad (i=1,2,\ldots,n) & \text{(F3)}\\
& -\alpha_i(R_0-R_i) \le x_i \le \alpha_i(R_0-R_i) \quad (i=1,2,\ldots,n) & \text{(F4)}\\
& -\alpha_i(R_0-R_i) \le y_i \le \alpha_i(R_0-R_i) \quad (i=1,2,\ldots,n) & \text{(F5)}\\
& \alpha_i \in \{0,1\} \quad (i=1,2,\ldots,n) & \text{(F6)}
\end{array}\right.
$$

In the above formulation, inequality (F2) forces the constraint that the circles have no overlap, and inequality (F3) says that the placed circles do not protrude from the container, and unplaced circles are centered at the origin. Inequalities (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. [4] applied a commercial nonlinear optimization 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 $0 \le \alpha_i \le 1$ 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 ($\mathrm{CPP1_R}(\delta)$)

$$
\sum_{i=1}^{n}\alpha_i(1-\alpha_i) \le \delta, \quad 0 \le \alpha_i \le 1 \ (i=1,2,\ldots,n).
$$

Note that when $\delta = 0$, this problem is equivalent to (CPP1). The heuristic method proposed in [4], initially sets $\delta = 0.05$, then solves a series of problems ($\mathrm{CPP1_R}(\delta)$) while reducing the value of δ by a fixed factor β. 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.

Algorithm 1 Heuristic algorithm by López et al. [4]

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 ($\mathrm{CPP1_R}(\delta)$) (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**.

3. Fundamentals of mixed integer DC programming

In this section, we briefly describe important properties of mixed integer DC programming. Throughout this section, we set $\infty = \infty - \infty$ for convention. Also, for any function $\phi : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$, we denote the effective domain of ϕ by $\operatorname{dom}\phi$, namely, $\operatorname{dom}\phi := \{x \in \mathbb{R}^n \mid \phi(x) < +\infty\}$.

3.1. Continuous DC programming

Definition 3.1. A real valued function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ 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 φ, 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.

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 3.2 ([8])**.** For DC functions f, f_i $(i = 1, \ldots, m)$, the functions defined by the following operations are also DC.

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

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

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

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

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

Here, we assume that $g, h : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ are closed proper convex functions satisfying $\emptyset \neq \operatorname{dom} g \subseteq \operatorname{dom} h$. DC programs are an important class of nonconvex optimization problems, with a wide range of applications. They have been studied since the late 20th century known to satisfy many nice mathematical properties such as *Toland-Singer duality*. Tao et al. [8] 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^*, the conjugate of g, namely, $g^*(\boldsymbol{y}) := \sup_{\boldsymbol{x} \in \mathbb{R}^n} \big(\boldsymbol{y}^\top \boldsymbol{x} - g(\boldsymbol{x})\big)$.

Proposition 3.3 ([8])**.** The following properties hold for any optimal solution $\boldsymbol{x}^*$ of (2).

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 $\inf_{\boldsymbol{y} \in \mathbb{R}^n} \big\{h^*(\boldsymbol{y}) - g^*(\boldsymbol{y})\big\}$

Definition 3.4. For a DC function $f = g - h$, a *stationary DC point* is vector $\boldsymbol{x}^*$ satisfying the condition

$$\partial g(\boldsymbol{x}^*) \cap \partial h(\boldsymbol{x}^*) \neq \emptyset. \tag{3}$$

In the case that both g and h are smooth, equation (3) is equivalent to the relation $\nabla g(\boldsymbol{x}^*) = \nabla h(\boldsymbol{x}^*)$, thus, a stationary DC point $\boldsymbol{x}^*$ will satisfy $\nabla f(\boldsymbol{x}^*) = \nabla g(\boldsymbol{x}^*) - \nabla h(\boldsymbol{x}^*) = 0$.

The DCA, due to Tao et al. [8], which finds a stationary point of the problem (2) is described in Algorithm 2.

Algorithm 2 DCA

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

Step 1: Choose $\boldsymbol{y}^{(k)} \in \partial h(\boldsymbol{x}^{(k)})$.

Step 2: Solve $\min_{x \in \mathbb{R}^n} g(x) - \big(h(x^{(k)}) + (y^{(k)})^\top (x - x^{(k)})\big)$ to obtain $x^{(k+1)}$.

Step 3: If $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\| = 0$, then output $\boldsymbol{x}^{(k+1)}$ and stop.
Otherwise set $k := k + 1$ and goto **Step1**.

Upon termination, this algorithm finds a stationary DC point, that is, an $\boldsymbol{x}^*$ satisfying (3).

We add here, that the above results can be applied to DC programs containing constraints defined by closed convex sets, through the introduction of indicator functions. More precisely, for the following constrained DC program:

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

where $S \subseteq \mathbb{R}^n$ is a closed convex set, we replace g by the function $g + \delta_S$, and remove the constraint $\boldsymbol{x} \in S$. Here, $\delta_S : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is the indicator function of S, defined by

$$\delta_S(\boldsymbol{x}) := \begin{cases} 0 & (\boldsymbol{x} \in S), \\ +\infty & (\boldsymbol{x} \notin S). \end{cases}$$

Obviously, the above constrained DC program is equivalent to the following program:

$$\underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}}\ f(\boldsymbol{x}) + \delta_S(\boldsymbol{x}) = \big(g(\boldsymbol{x}) + \delta_S(\boldsymbol{x})\big) - h(\boldsymbol{x}).$$

3.2. Mixed integer DC programming

We now consider mixed integer DC programs, which are DC programs in which some of the variables are restricted to integer values. A mixed integer DC program can be formulated as follows:

$$\left| \begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & f(\boldsymbol{x}) = g(\boldsymbol{x}) - h(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} = (\boldsymbol{x}_M, \boldsymbol{x}_N) \in S \\ & \boldsymbol{x}_M \in \mathbb{R}^M, \boldsymbol{x}_N \in \mathbb{Z}^N. \end{array} \right. \tag{4}$$

Here, $n = |M|+|N|$, $S \subseteq \mathbb{R}^n$ is a closed convex set, and all the other settings are same as in problem (2). Solving this problem usually involves the use of some type of continuous relaxation. The obvious relaxation which replaces $\boldsymbol{x} \in \mathbb{R}^M \times \mathbb{Z}^N$ 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 3.5. Let φ be a function $\varphi : \mathbb{R}^M \times \mathbb{Z}^N \to \mathbb{R} \cup \{+\infty\}$. If there exists a (continuous) closed convex function $\hat{\varphi} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ satisfying the condition

$$\hat{\varphi}(\boldsymbol{x}) = \varphi(\boldsymbol{x}) \quad (\boldsymbol{x} \in \mathbb{R}^M \times \mathbb{Z}^N), \tag{5}$$

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

In the case that φ is originally defined as the restriction of a continuous convex function g to $\mathbb{R}^M \times \mathbb{Z}^N$, φ is trivially convex extensible with the obvious convex extension 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 3.6. For a function $\psi : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$, the *epigraph* of ψ is the set $\mathrm{epi}_{\mathbb{R}^n}\psi := \{(\boldsymbol{x}, x_{n+1}) \mid x_{n+1} \geq \psi(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^n\} \subseteq \mathbb{R}^{n+1}$. The epigraph $\mathrm{epi}_{\mathbb{R}^M \times \mathbb{Z}^N}\psi$ of a discrete function $\psi : \mathbb{R}^M \times \mathbb{Z}^N \to \mathbb{R} \cup \{+\infty\}$ is defined analogously, by replacing $\boldsymbol{x} \in \mathbb{R}^n$ with $\boldsymbol{x} \in \mathbb{R}^M \times \mathbb{Z}^N$.

For a convex extensible function $\varphi : \mathbb{R}^M \times \mathbb{Z}^N \to \mathbb{R} \cup \{+\infty\}$, the *closed convex hull extension* $\varphi^{\mathrm{cl}} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is the continuous closed convex function whose epigraph $\mathrm{epi}_{\mathbb{R}^n}\varphi^{\mathrm{cl}}$ is equal to the convex closure of $\mathrm{epi}_{\mathbb{R}^M \times \mathbb{Z}^N}\varphi$.

The next theorem shown by Okuno et al. [6], is a generalization of Theorem 3 of Maehara et al. [5] for pure discrete functions defined on $\mathbb{Z}^n$. It says that continuous relaxations using closed convex hull extensions have no integrality gap.

Theorem 3.7 ([6])**.** Let $\varphi_1, \varphi_2 : \mathbb{R}^M \times \mathbb{Z}^N \to \mathbb{R} \cup \{+\infty\}$ be convex extensible discrete functions such that $\emptyset \neq \mathrm{dom}\, \varphi_1 \subseteq \mathrm{dom}\, \varphi_2$. Then,

$$\inf\left\{\varphi_1(\boldsymbol{x}) - \varphi_2(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathbb{R}^M \times \mathbb{Z}^N\right\} = \inf\left\{\varphi_1^{\mathrm{cl}}(\boldsymbol{x}) - \hat{\varphi}_2(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathbb{R}^{M+N}\right\} \tag{6}$$

for any closed convex extension $\hat{\varphi}_2$ of φ_2. Thus, we can solve (4) by solving a continuous relaxation in which φ_1 is replaced by its closed convex hull extension (and φ_2 by any closed convex extension).

Based on the above theorem and the DCA of [8], Okuno et al. [6] proposed the following Mixed Integer DCA(MIDCA) for the mixed integer DC program (4). Again, this is an extension of the algorithm of Maehara et al. [5] for discrete DC programs.

Algorithm 3 MIDCA

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

Step 1: Choose $\boldsymbol{y}^{(k)} \in \partial h(\boldsymbol{x}^{(k)})$.

Step 2: Find a solution $\boldsymbol{x}^{(k+1)}$ of the following subproblem:

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

Step 3: If $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\| = 0$, 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 (4) at $\boldsymbol{x}^{(k)}$. Thus, each $\boldsymbol{x}^{(k)}$ is always feasible to (4). Furthermore, the objective value strictly decreases every iteration. Upon termination, the algorithm outputs a DC stationary point satisfying (3) with g replaced by $\left(g + \delta_S\right)^{\mathrm{cl}}$.

3.3. Advantages and limitations of the MIDCA

Before proceeding to the next section, we make some remarks on the advantages of the MIDCA together with its limitations.

Advantages: The attractiveness of the MIDCA is its theoretical aspect. As mentioned in the previous section, the algorithm improves the objective value while retaining feasibility, in every iterarion. Moreover, like many continuous optimization algorithms, there is a theoretical guarantee that any cluster point of a generated sequence satisfies the optimality condition for the mixed integer DC program under consideration. In the practical aspect, by starting from a good initial point (for example, a feasible solution output by another mixed integer nonlinear solver), the MIDCA is able to find a superior solution unless the given initial point is a DC stationary point.

Limitations: On the other hand, the MIDCA tends to be numerically expensive, since it solves a sequence of mixed integer convex programs. Particularly, the performance of the MIDCA significantly relies on the solver used for solving mixed integer

convex subproblems. However, with help from fast solvers such as those in SCIP and Gurobi, the MIDCA works effectively for mixed integer DC programs.

4. A new heuristic algorithm for the circle packing problem

In this section, we propose a heuristic algorithm for the circle packing problem.

In doing so, we first reformulate (CPP1) as a mixed integer DC program, then modify some variables and constraints. In its original form, the algorithm alternately solves two problems; one is the modified MIDC reformulation of (CPP1), which we solve by the MIDCA, and use to decide the tentative positions of the circles as well as those that should be placed, and the other is a related problem, in which we repack *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 unplaced circles into positions which can be used to improve it.

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

4.1. Reformulation 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 βij. More explicitly, we replace the quadratic term $\alpha_i\alpha_j$ by β_{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 constraint (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, $P_{ij} > 0$ denotes a penalty parameter for each i, j. 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 f. We now express the objective function as a DC function.

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

This function contains the max operator, and hence is not smooth. We add auxiliary variables u_{ij} $(1 \le i < j \le n)$ to deal with this. The resulting problem, which we call $(\mathrm{CPP2_{DC}})$, can be described as below.

$$(\mathrm{CPP2_{DC}}) \left| \begin{array}{ll} \underset{\boldsymbol{x},\boldsymbol{y},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{u}}{\text{minimize}} & \\ \quad -\pi\sum_{i=1}^{n}\alpha_i R_i^2 + \sum_{1\le i<j\le n} P_{ij}u_{ij} - \sum_{1\le i<j\le n} P_{ij}\{(x_i-x_j)^2+(y_i-y_j)^2\} & \text{(F7)} \\ \text{subject to} \quad (x_i-x_j)^2+(y_i-y_j)^2 \le u_{ij} \quad (1\le i<j\le n) & \text{(F2e)} \\ \quad \beta_{ij}(R_i+R_j)^2 \le u_{ij} \quad (1\le i<j\le n) & \text{(F2f)} \\ \quad \text{constraints (F3),(F4),(F5),(F2b),(F2c),(F2d),(F6)} & \end{array} \right.$$

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 initial values of (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 shows 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 constraint (F3) forces all unplaced 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 unplaced circles to be situated at the origin, we alter constraint (F3) as below, and allow unplaced circles to be placed anywhere within the container.

$$x_i^2 + y_i^2 \le (R_0 - R_i)^2 \quad (i = 1, 2, \ldots, n) \qquad \text{(F3a)}$$

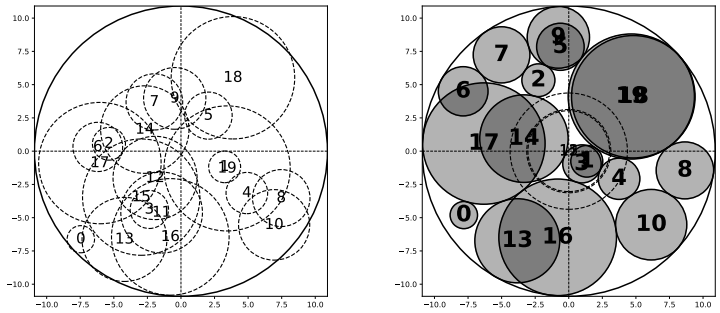

Figure 1: Solution of ($\mathrm{CPP2_{DC}}$): Initial solution ⇒ MIDC stationary point

The resulting problem which we will call ($\mathrm{CPP3_{DC}}$), can be explicitly written as below.

$$
(\mathrm{CPP3_{DC}})\quad
\left|\begin{array}{lll}
\underset{\boldsymbol{x},\boldsymbol{y},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{u}}{\text{minimize}} & & \\
\quad -\pi\displaystyle\sum_{i=1}^{n}\alpha_i R_i^2 + \sum_{1\le i<j\le n} P_{ij}u_{ij} - \sum_{1\le i<j\le n} P_{ij}\{(x_i-x_j)^2+(y_i-y_j)^2\} & & \text{(F7)}\\
\text{subject to} \quad (x_i-x_j)^2+(y_i-y_j)^2 \le u_{ij} & (1\le i<j\le n) & \text{(F2e)}\\
\qquad\qquad \beta_{ij}(R_i+R_j)^2 \le u_{ij} & (1\le i<j\le n) & \text{(F2f)}\\
\qquad\qquad x_i^2+y_i^2 \le (R_0-R_i)^2 & (i=1,2,\ldots,n) & \text{(F3a)}\\
\qquad\qquad \alpha_i+\alpha_j-1 \le \beta_{ij} & (1\le i<j\le n) & \text{(F2b)}\\
\qquad\qquad 0\le \beta_{ij}\le \alpha_i & (1\le i<j\le n) & \text{(F2c)}\\
\qquad\qquad 0\le \beta_{ij}\le \alpha_j & (1\le i<j\le n) & \text{(F2d)}\\
\qquad\qquad \alpha_i\in\{0,1\} & (i=1,2,\ldots,n) & \text{(F6)}
\end{array}\right.
$$

In Figure 2, we show the results of applying the MIDCA to ($\mathrm{CPP3_{DC}}$), 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.

4.2. 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

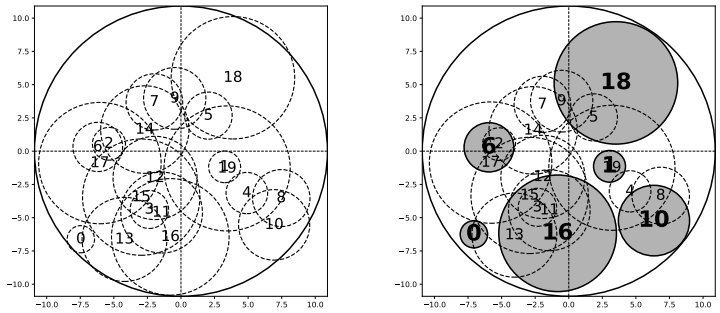

Figure 2: Solution of ($\mathrm{CPP3_{DC}}$): Initial solution ⇒ MIDC stationary point

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 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.

$$\text{(OVERLAP1)} \left| \begin{array}{ll} \underset{\boldsymbol{x},\boldsymbol{y}}{\text{minimize}} & \displaystyle\sum_{1\le i<j\le n} \max\{0, (R_i+R_j)^2 - (x_i-x_j)^2 - (y_i-y_j)^2\} \\ \text{subject to} & x_i^2 + y_i^2 \le (R_0 - R_i)^2 \qquad (i = 1, 2, \ldots, n) \end{array} \right.$$

The objective function of the above problem is both nonsmooth and nonconvex. By using the same techniques as in Section 4.1, we obtain the following DC program, which we call ($\mathrm{OVERLAP1_{DC}}$).

$$\text{(OVERLAP1}_{\mathrm{DC}}\text{)} \left| \begin{array}{lll} \underset{\boldsymbol{x},\boldsymbol{y},\boldsymbol{u}}{\text{minimize}} & \displaystyle\sum_{1\le i<j\le n} u_{ij} - \sum_{1\le i<j\le n} \{(x_i-x_j)^2 + (y_i-y_j)^2\} & \\ \text{subject to} & (x_i-x_j)^2 + (y_i-y_j)^2 \le u_{ij} & (1 \le i < j \le n) \\ & (R_i+R_j)^2 \le u_{ij} & (1 \le i < j \le n) \\ & x_i^2 + y_i^2 \le (R_0 - R_i)^2 & (i = 1, 2, \ldots, n) \end{array} \right.$$

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

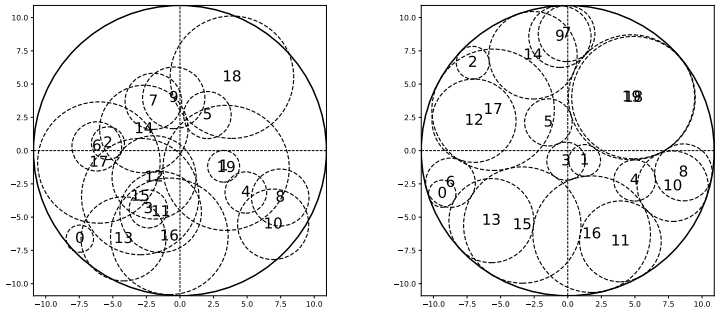

Figure 3: Solution of (OVERLAP1): Initial solution ⇒ DC stationary point

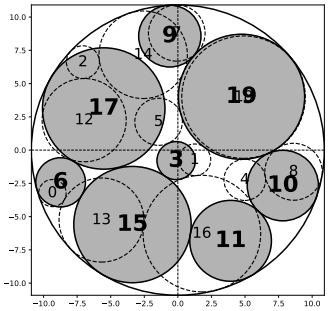

Figure 4: Solution of ($\mathrm{CPP3_{DC}}$) (Initial solution: DC stationary point of Fig. 3)

4.3. Basic framework of our algorithm

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 ($\mathrm{OVERLAP1_{DC}}$) in which the positions of circles already placed are 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) $(i \in S)$, 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). Specifically, this problem, which is named (OVERLAP2($\bar{z}$)), is described as follows:

$$(\text{OVERLAP2}(\bar{z})) \left| \begin{array}{ll} \underset{\boldsymbol{x},\boldsymbol{y}}{\text{minimize}} & \displaystyle\sum_{i \in S} \sum_{j \in T} \max\{0, (R_i + R_j)^2 - (x_i - \bar{x_j})^2 - (y_i - \bar{y_j})^2\} \\ \text{subject to} & x_i^2 + y_i^2 \leq (R_0 - R_i)^2 \qquad (i \in S) \end{array} \right. \tag{7}$$

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 unplaced circles will move to positions having little overlap with the placed circles, allowing us to newly place them. As with (OVERLAP1), we solve (OVERLAP2($\bar{z}$)) to obtain a stationary point 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 marked 0,1,2, and 4, indicated by dotted lines, moving so as to reduce overlap with the placed circles. Using the resulting configuration as the initial solution to problem (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.

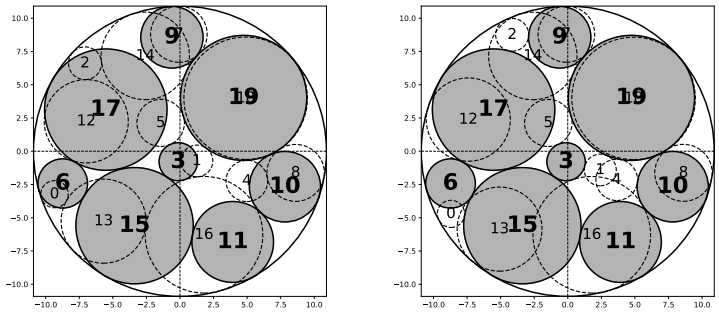

Figure 5: Solution of (OVERLAP2) : Initial (Fig. 2) ⇒ stationary point

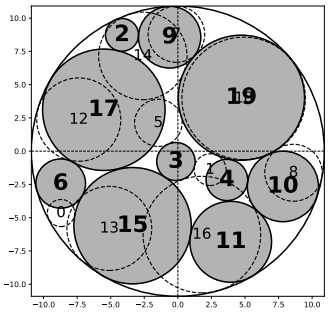

Figure 6: MIDC point of ($\mathrm{CPP3_{DC}}$): (Initial solution: Fig 5)

Basically, our algorithm iterates this procedure until no further improvement occurs. The total framework can be formally described as in Algorithm 4. Here, $f_{\mathrm{DC}}(\boldsymbol{z})$ denotes the objective function (F7) of problem ($\mathrm{CPP3_{DC}}$).

Algorithm 4 Basic framework of our proposed algorithm

Step 0: Generate an initial solution by the following procedure.

Step 0-1 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 $k := 0$.

Step 0-2 Solve ($\mathrm{OVERLAP1_{DC}}$) 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 $\boldsymbol{z}^{(0)} = \boldsymbol{z}'$.

Step 1: Apply the MIDC algorithm to ($\mathrm{CPP3_{DC}}$), and find an MIDC stationary point $\boldsymbol{z}^{(k+1)} := (\boldsymbol{x}^{(k+1)}, \boldsymbol{y}^{(k+1)}, \boldsymbol{\alpha}^{(k+1)}, \boldsymbol{\beta}^{(k+1)}, \boldsymbol{u}^{(k+1)})$.

Step 2: If $f_{\mathrm{DC}}(\boldsymbol{z}^{(k)}) = f_{\mathrm{DC}}(\boldsymbol{z}^{(k+1)})$, then output $\boldsymbol{z}^{(k+1)}$ and stop.

Step 3: Solve the problem OVERLAP2($\boldsymbol{z}^{(k+1)}$) to find a 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 **Step 1**.

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.

4.4. A modified version

In Step 1 of Algorithm 4, we solve (CPP3$_{\rm DC}$) with the MIDC algorithm. However, use of the MIDC is quite time-consuming, since it requires solving a sequence of mixed integer quadratic optimization problems with quadratic constraints. In this section, we propose an alternate method for solving (CPP3$_{\rm DC}$). Instead of solving it as a mixed integer DC program by the MIDCA, 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,

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

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 continuous variables. In our algorithm, we take advantage of this fact to speedup Step 1 (solving (CPP3$_{\rm DC}$)) of our 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.

The modification of Step 1 can be formally described as follows (Steps 0, 2, and 3 are unchanged from Algorithm 4).

Algorithm 5 Modification of **Step 1** in Algorithm 4

Step 1: ($\boldsymbol{z}^{(k)} = (\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, \boldsymbol{\alpha}^{(k)}, \boldsymbol{\beta}^{(k)}, \boldsymbol{u}^{(k)})$ 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.

Step1.1: Fix $(\boldsymbol{x}, \boldsymbol{y}) = (\bar{\boldsymbol{x}}^{(j)}, \bar{\boldsymbol{y}}^{(j)})$ in (CPP3$_{\rm DC}$), and use a MIP solver to find $(\bar{\boldsymbol{\alpha}}^{(j+1)}, \bar{\boldsymbol{\beta}}^{(j+1)}, \bar{\boldsymbol{u}}^{(j+1)})$ by solving (CPP4).

Step1.2: 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 $(\bar{\boldsymbol{x}}^{(j+1)}, \bar{\boldsymbol{y}}^{(j+1)}, \bar{\boldsymbol{u}}^{(j+1)})$ by solving (OVERLAP3$_{\rm DC}$). Set $\bar{\boldsymbol{z}}^{(j+1)} := (\bar{\boldsymbol{x}}^{(j+1)}, \bar{\boldsymbol{y}}^{(j+1)}, \bar{\boldsymbol{\alpha}}^{(j+1)}, \bar{\boldsymbol{\beta}}^{(j+1)}, \bar{\boldsymbol{u}}^{(j+1)})$.

Step1.3: 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 **Step2**, otherwise set $j := j + 1$ and goto **Step1.1**.

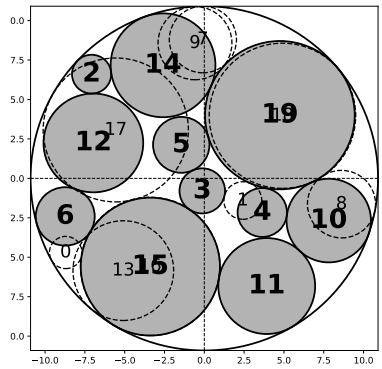

Figure 7: Final solution

5. Numerical experiments

In this section, we report the results of numerical experiments. The computer we used is a Windows 10, 4.00GHz PC with Intel i7-6700K core and 16GB memory. All implementations were coded with Python 3.6, and the solver we used is Gurobi 7.5.2 (both MIP and nonlinear optimization). For our problem data, we used the circle packing problems originally used in [4], and available from the OR-Library [7]. Our experiments consist of two sets; in the first we compared our new algorithms Algorithms 4 and 5 (original and modified versions) with the existing method of López et al. [4]. We add here, that in solving OVERLAP2($\boldsymbol{z}^{(k+1)}$) in Step 3, we converted it to DC form, and

applied the DCA, to obtain a DC stationary point. Results showed that our modified method showed most promise, thus, in the next set, we conducted more experiments on this algorithm in depth.

5.1. The first set of experiments: comparison of algorithms

In order to investigate the efficiency and solution quality of the two versions of our new algorithms, we compared them with the algorithm of López et al. [4] as well as direct application of SCIP [1] to (CPP1) and the MIDCA to (CPP3_{DC}). The test problems we used are the same as those used in [4]. 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, n-2, and n-3 for each $n = 10, 20, 30$. 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. For our two algorithms, the values of the penalty parameters P_{ij} $(1 \leq i < j \leq n)$ were set to $P_{ij} = 3(R_i^2 + R_j^2)/(R_i + R_j)^2$ for each i, j. 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 [4] (for example, the best value found by our implementation for the data set $10-1$ was 26.45, whereas the best value reported in [4] is 83.08) . Thus, in the interest of fairness, for this algorithm we have reproduced the best value and CPU time results from [4] (CPU times are the values obtained by dividing the total time in Table 1 of [4] by the number of tries, five, and are shown in parentheses, as computing environments are different). We mention that all solutions found by our algorithms contained no overlap, and were feasible to the original problems, as are those published in [4], thus the best values shown in Table 1 are precisely the total area of the circles placed in the container, for each problem instance. In Table 1, abbreviations are as follows.

SCIP: Results of directly solving (CPP1) with SCIP5.0.0,

López: Results as reported in [4], of solving (CPP1) with the algorithm of López et al.,

MIDC: Results of solving (CPP3_{DC}) by the MIDCA,

Method 1: Results of solving (CPP3_{DC}) by the original version of our method,

Method 2: Results of solving (CPP3_{DC}) by the modified version of our method.

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	(115.4)	0.919	2.945	1.606
	best value	83.080	83.080	77.455	83.080	83.080
10-2	avg. time	15.650	(3.4)	0.840	4.803	2.297
	best value	124.620	124.620	110.140	120.956	120.956
10-3	avg. time	2596.380	(783.5)	0.586	2.423	0.660
	best value	197.071	197.071	138.590	193.757	193.757
20-1	avg. time	3600	(970.2)	20.569	162.322	6.297
	best value	105.480	143.755	101.184	141.139	141.139
20-2	avg. time	3600	(1032.8)	23.754	76.034	5.960
	best value	117.678	219.065	176.508	215.571	218.086
20-3	avg. time	3600	(1137.8)	27.064	61.940	10.661
	best value	211.069	290.506	207.758	295.617	295.617
30-1	avg. time	3600	(1909.8)	338.404	1437.161	21.498
	best value	210.236	254.244	184.695	250.774	250.774
30-2	avg. time	3600	(1517)	680.300	2071.709	28.850
	best value	232.727	372.064	261.090	376.204	377.217
30-3	avg. time	3600	(1409.2)	551.343	1824.820	51.066
	best value	261.206	502.017	324.590	493.377	493.377

Obviously, simply using the MIDCA on (CPP3_{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 [4] is slightly better than ours, but not by much. As to our two methods, the modified version is clearly superior, in both that it is much faster, and the solutions quality is at least as good as the original, and sometimes better.

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

5.2. 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 modified 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 and CPU times for the López et al. algorithm are again, the results of 5 initial solutions, taken from [4]. We notice that for all data, our method was able to find better solutions, within reasonable computation time.

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

data	results	López	Method 2	data	results	López	Method 2
30-1	avg. time	(1909.8)	21.648	40-1	avg. time	(1739.4)	83.814
	best value	254.244	254.477		best value	318.914	340.589
30-2	avg. time	(1517)	32.344	40-2	avg. time	(1739.4)	137.065
	best value	372.064	383.863		best value	486.941	507.734
30-3	avg. time	(1409.2)	47.031	40-3	avg. time	(2717)	202.583
	best value	502.017	512.127		best value	642.909	680.293

Next, to determine the practical 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 [4] 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 [4], we show results only for our method.

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

While there is no accurate way to evaluate the quality of the solutions, drawings of the actual configurations shown in Figures 8 and 9 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.

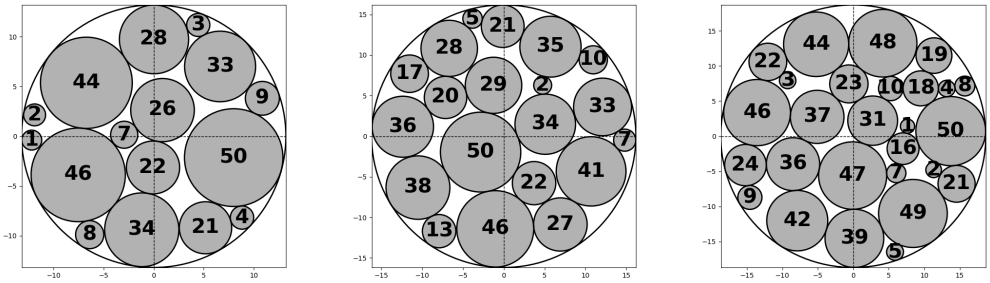

Figure 8: Best solutions for $n = 50$ circles: (left) 50-1, (center) 50-2, (right) 50-3

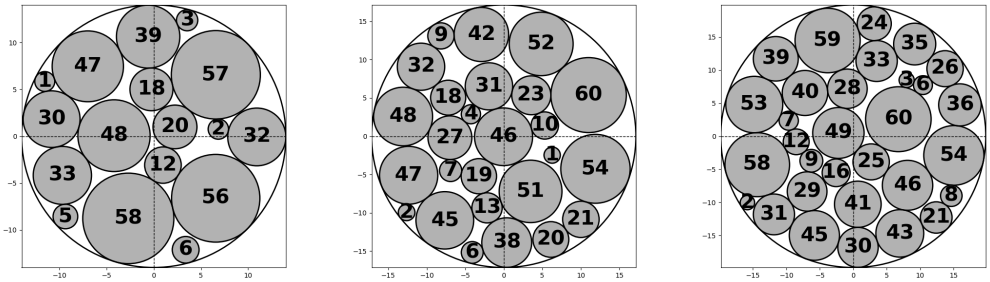

Figure 9: Best solutions for $n = 60$ circles: (left) 60-1, (center) 60-2, (right) 60-3

6. 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 and proposed a heuristic 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.

Acknowledgements

This work was partially supported by JSPS Grants-in-Aid for Young Scientists 15K15943.

References

[1] Achterberg, T. (2009). SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1-1, 1–41.

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

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

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

[5] Maehara, T, Marumo, N. & Murota, K. (2018). Continuous relaxation for discrete DC programming. *Mathematical Programming, Series B*, 169. 199–219.

[6] Okuno, T. & Ikebe, Y. T. (2017). 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.

[7] OR-Library. http://people.brunel.ac.uk/˜mastjjb/jeb/orlib/circleinfo.html/ accessed 20 December 2018.

[8] Tao, P. D. & An, L. T. H (1997). Convex analysis approach to dc programming. *Acta Mathematica Vietnamica*, f22-1, 289–355.