Article

Balanced Circular Packing Problems with Distance Constraints

Tetyana Romanova 1,2, Olexandr Pankratov 1,2, Igor Litvinchev 3, Petro Stetsyuk 4, Oleksii Lykhovyd 4, Jose Antonio Marmolejo-Saucedo 5,6 and Pandian Vasant 7

Abstract: The packing of different circles in a circular container under balancing and distance conditions is considered. Two problems are studied: the first minimizes the container’s radius, while the second maximizes the minimal distance between circles, as well as between circles and the boundary of the container. Mathematical models and solution strategies are provided and illustrated with computational results.

Keywords: dense circular packing; sparse circular packing; balance; mathematical models; nonlinear optimization; Shor’s r-algorithm

1. Introduction

The circular packing problem (CPP) consists of placing several circles in a larger object called a container. The circles must be arranged without mutual overlapping (the non-overlapping condition) and completely inside the container (the containment condition). CPPs (including CPPs in non-Euclidean geometry) form a broad category at the boundary between computational geometry and combinatorics [1–3]. The optimized CPP aims to find “the best” layout, e.g., maximizing the space utilization of the container or minimizing the size of the container. Optimized CPPs have multiple applications in logistics and production planning, biology and medicine, chemistry and additive manufacturing [4–6].

Various types of CPP can differ according to circle sizes (different or equal) [4,7–9] and container shapes (circles or polyhedrons) [10–12]. In open-dimension CPPs, the shape of the container is typically fixed, while the size is optimized [13,14]. The problem of packing circles into containers with prohibited zones is considered in [15–18]. Prohibited zones typically result in non-convex, multiconnected and/or disconnected containers.

Additional constraints motivated by applications can be introduced. For example, in a satellite module layout design, equilibrium/balance constraints are imposed to assure the proximity of the gravity center of the container to the corresponding center of the
circles [19–23]. In many applications, CPPs are formulated for circles defined in non-Euclidean norms [17,24,25]. Nesting, i.e., the stacking of circular disks one over another, can also be considered [26–30].

Solution approaches to CPPs can be roughly divided into four large classes [4,6,31]. The techniques of the first group are based on nonlinear programming formulations of the CPP, and these approaches apply different exact or approximation methods for this class of optimization problems [5,11,12,15,20,21]. The second class of approaches is based on tessellating or approximating the container with a finite grid [24,26,32–34]. Then, the CPP is approximately reduced to a large-scale discrete programming problem in order to assign the centers of the circles to grid nodes. Using metaheuristics techniques for CPPs forms the third class of approaches [35,36]. The last group is represented by hybrid approaches combining the algorithmic ideas of the previous groups [7,8,10,13,14,19,28,31]. A collection of benchmark problem instances and corresponding best-known solutions is presented on E. Specht’s website [37], specifically for the problem of packing circles into various containers.

The objective in most optimized packing problems is to find the densest possible packing arrangement, e.g., maximizing the occupation of the container or finding the minimal container for an open-dimension problem. However, in many up-to-date applications, the dense packing concept must be revised.

In sparse packing [38], the distance between the objects, as well as between the objects and the container, must be sufficiently large. In container-loading applications, these conditions arise to facilitate access to the objects during loading/unloading. Generating lightweight void structures by 3D printing requires sufficient distance between the voids to assure the mechanical strength of the part [39]. Sparse packing arises in thermal deburring, which is a state-of-the-art technology for cleaning 3D-printed parts of non-sintered powder particles [38]. The burrs are removed from 3D objects by exploding gas mixtures in a closed chamber. To achieve a reasonable processing quality and a “uniform” distribution of power and thermal effects, the parts must be placed at sufficient distances from one another.

In this work, the packing of different circles in a circular container under balancing and distance conditions is considered. Dense and sparse formulations are studied. The first problem aims to minimize the container’s radius. In the second, the minimal distance between circles, and between the circles and the container’s boundary, is maximized.

### 2. Dense Packing Circular Problem with Distance and Balancing Conditions

In this section, a circular packing problem is studied under balancing and distance conditions. The objective is to obtain the smallest possible circular container (dense packing).

Let $S_i, i = 1, 2, \ldots, n$ be a family of circles with radii $r_i$ and weights $w_i$. Let $S$ be an external circle (container) centered at the origin and having radius $r$. By the definition, the center of gravity of the circle $S$ is also located at the origin. Denote by $(x_i, y_i)$ the unknown center coordinates of $S_i$, and let $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$.

Let $\rho_{ij} > 0$ be the minimum permitted distance between $S_i$ and $S_j$, $1 \leq i < j \leq n$. In addition, let $\rho_i > 0$ be the minimum permitted distance between $S_i$ and the boundary of the object $S^* = R^2 \setminus \text{int } S$, $i = 1, \ldots, n$. More specifically, let

$$\text{dist}(S_i(v_i), S_j(v_j)) \geq \rho_{ij}, \ 1 \leq i < j \leq n,$$

$$\text{dist}(S_i(v_i), S^*) \geq \rho_i, \ i = 1, \ldots, n$$

where $\text{dist}(A,B) = \min_{a\in A,b\in B} \text{dist}(a,b)$ and $\text{dist}(a,b)$ refer to the (Euclidean) distance between two points $a$ and $b$.

A dense circular balanced packing (DCBP) problem can be stated as follows. Pack the circles $S_i, i = 1, 2, \ldots, n$ in a minimum-radius circular container $S$ subject to the distance
constraints in (1), such that the gravitation center of the family of circles \( S_i, \ i = 1, 2, \ldots, n \) is located at the origin (i.e., coincides with the gravity center of \( S \)).

A trivial lower bound on the minimal radius of \( S \) is

\[
    r_{\text{low}} = \max_{i=1, \ldots, n} r_i + \rho, \rho = \max\{ \max_{i=1, \ldots, n} \rho_i, 0.5 \max_{1 \leq i < j \leq n} \rho_{ij} \}
\]

The DCBP is stated as a nonlinear optimization problem in \( \mathbb{R}^{2n+1} \):

\[
    r^* = \min_{r,x,y} r
\]

subject to

\[
    (x_i - x_j)^2 + (y_i - y_j)^2 \geq (r_i + r_j + \rho_{ij})^2, \quad 1 \leq i < j \leq n, \quad (3)
\]

\[
    x_i^2 + y_i^2 \leq (r - r_i - \rho_i)^2, \quad i = 1, \ldots, n, \quad (4)
\]

\[
    \sum_{i=1}^{n} \lambda_i x_i = 0, \quad \sum_{i=1}^{n} \lambda_i y_i = 0, \quad (5)
\]

\[
    r \geq r_{\text{low}} \quad (6)
\]

where \( \lambda_i = w_i / \sum_{j=1}^{n} w_j \).

The constraints in (3) guarantee that the distance between the pair of circles \( S_i \) and \( S_j \) is at least \( \rho_{ij}, 1 \leq i < j \leq n \), while the constraints in (4) assure that the distance between \( S_i \) and the boundary of \( S \) is at least \( \rho_i, i = 1, 2, \ldots, n \). The linear constraints in (5) state that the gravity center of the family \( S_i, i = 1, 2, \ldots, n \) coincides with the origin, while the “dummy” constraint in (6) provides the lower bound for the objective value.

2.1. Sequential Algorithm for DCBPs

Using penalty functions, the problem in (1)–(5) can be transformed into the unconstrained minimization of a non-smooth function

\[
    \min_{r,x,y} \{ f(r,x,y) = r + \Phi_p(r,x,y) \}
\]

where the penalty function \( \Phi_p(r,x,y) \) has the form

\[
    \Phi_p(r,x,y) = P_1 F_1(r,x,y) + P_2 F_2(x,y) + P_3 \max\{ 0, -r + r_{\text{low}} \} \quad (8)
\]

Here \( P = \{ P_1, P_2, P_3 \} \), where the penalization coefficients \( P_k, k = 1, 2, 3 \) are positive, and the functions \( F_1(r,x,y) \) and \( F_2(x,y) \) are as follows:

\[
    F_1(r,x,y) = \sum_{i=1}^{n} \max\{ 0, x_i^2 + y_i^2 - (r - r_i - \rho_i)^2 \}
\]

\[
    \quad + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \max\{ 0, - (x_i - x_j)^2 - (y_i - y_j)^2 + (r_i + r_j + \rho_{ij})^2 \} \quad (9)
\]

\[
    F_2(x,y) = \max\{ 0, \sum_{i=1}^{n} \lambda_i x_i, - \sum_{i=1}^{n} \lambda_i x_i \} + \max\{ 0, \sum_{i=1}^{n} \lambda_i y_i, - \sum_{i=1}^{n} \lambda_i y_i \} \quad (10)
\]

The local minimization in (2)–(6) is equivalent to the local minimization in the unconstrained non-smooth problem in (7)–(10). If \( \Phi_p(r,x,y) = 0 \) in the local minimum of the function \( f(r,x,y) \) for certain values of the coefficients \( P = \{ P_1, P_2, P_3 \} \), then this point is also a local minimum for the problem in (2)–(6). The choice of the penalty coefficients \( P_1, P_2 \) and \( P_3 \) allows the adjustment of the violation of the constraints in (3)–(6). The coefficient \( P_1 \) corresponds to the “total violation” of the quadratic constraints in (3)–(4), \( P_2 \) “controls” the linear constraints in (5), while \( P_3 \) adjusts the constraint in (6).
The sequential algorithm to obtain the best local solution to the problem in (2)–(6) uses the multistart method with randomly selected starting points and is based on a variant of Shor’s r-algorithm (see, e.g., [40–43] and Appendix A) to locally minimize \( f(r, x, y) \). The feasible starting points for the problem in (2)–(6) are not necessary. This algorithm can also be used for the particular case of (2)–(6), where constraints on the center of gravity of the system of circles are eliminated. It is sufficient to set \( P_z = 0 \); that it is equivalent to excluding the constraints in (4) from the problem in (2)–(6).

To present the sequential algorithm, denote by \( n_{starts} \) the number of starting points uniformly distributed inside the circular container of radius \( r_{up} \) (the best upper bound for the container’s radius). At the beginning of the process, \( r_{up} = \sum_{i=1}^{n_{starts}} (r_i + \rho) \), \( \rho = \max \{ \max \rho_i, 0.5 \max \rho_{ij} \} \). If the value of the local minimum decreases, then \( r_{up} \) is sequentially updated for all starting points. To obtain the local minimum of \( f(r, x, y) \), the r-algorithm with an adaptive step-size and fixed space dilation [41] is used. The best local minimum of the function \( f(r, x, y) \), corresponding to the penalty \( \Phi_p(r, x, y) \) close to zero, is considered a solution to the problem in (2)–(6). This way, the best value \( r_{up} \) of the radius is obtained, together with the coordinates \((x^{up}, y^{up})\) of the circles.

2.2. Sequential Implementation of a Parallel Algorithm for DCBP

In a parallel algorithm, multiple searches for local solutions are implemented using the modification of the r-algorithm. This differs from a sequential approach that starts only one search. The parallel algorithm is implemented using the \((k + 1)\)-processor “master–slave” technique. One processor is selected as the “main” (Master), while the others are “subordinate” (Slave).

In the Master processor, \( k \) starting points are randomly generated in a circle with a radius \( r_{up} = \sum_{i=1}^{n_{starts}} (r_i + \rho) \), \( \rho = \max \{ \max \rho_i, 0.5 \max \rho_{ij} \} \). Then, these points are sent to the Slave processors. Each Slave processor starts the local minimization of the function \( f(r, x, y) \) from its starting point. When the r-algorithm terminates on any Slave processor, the results are transferred to the Master processor. As soon as the local minimum of the problem in (2)–(6) is obtained, the radius of the container must be compared with the best value \( r_{up} \). If the radius is smaller than \( r_{up} \), the value of \( r_{up} \) is updated correspondingly, and the values \((x^{up}, y^{up})\) of the coordinates of the centers of the circles are saved. Then, the Master processor selects a new starting point, which is passed to the Slave processor on which the r-algorithm just finished to begin the new search for the local minimum. The process stops either when all starting points have been investigated or the computational time limit has been exceeded.

The parallel algorithm is implemented using the C++ programming language and the MPI parallel programming environment. The standard rand function is used to generate pseudo-random numbers. For local minimization, the module ralg05 is used, corresponding to the Octave code [44] of the modification of the r-algorithm presented in [41]. The program is coded to work for a cluster in an MPI environment and on a Linux operating system. It can be executed on a single processor or on several processors in parallel.

Here, the cluster SKIT-4 of the V.M. Glushkov Institute of Cybernetics [45] is used. The speedup of the parallel algorithm for the DCBP problem is studied for an instance with 50 circles: 20 circles of radius 10, 15 circles of radius 20, 10 circles of radius 30 and 5 circles of radius 40. It is assumed that \( w_i = r_i \), \( i = 1, 2, \ldots, n \). The multistart strategy is implemented with 5000 starting points, and the number of processors ranges from 1 to 32. The results are presented in Table 1. Here, \( k \) denotes the number of processors, \( t_{up} \) is the problem solution time in seconds, \( C_k \) is the parallel algorithm speedup and \( E_k \) is the parallel algorithm efficiency. The parallel speedup is calculated by the formula \( C_k = t_{up}/t_k \), \( k = 1, 4, 8, 12, 16, 24, 32 \), where \( t_0 \) is the solution time for the serial algorithm with one processor, and \( t_k \) is the solution time for the parallel algorithm with \( k \) processors. The effect of parallelization is estimated as \( E_k = C_k/k \). It follows from Table 1 that
for 4 processors the solution time is reduced by 2.95 times, for 8 - 5.78 times, for 16 - 13.43 times, for 24 - 14.31 times and for 32 - 15 times.

Table 1. Solution time, speedup and efficiency for a DCBP problem instance with 50 circles solved on the cluster SKIT-4.

| $k$ | $t_k$  | $C_k$  | $E_k$  |
|-----|--------|--------|--------|
| 1   | 2159.68| 1.00   | 1.00   |
| 4   | 731.08 | 2.95   | 0.74   |
| 8   | 373.85 | 5.78   | 0.72   |
| 12  | 209.04 | 10.33  | 0.86   |
| 16  | 160.79 | 13.43  | 0.84   |
| 24  | 150.93 | 14.31  | 0.60   |
| 32  | 144.02 | 15.00  | 0.47   |

Figure 1 presents the solution time compared to the number of processors used, Figure 2 shows the speedup diagram and Figure 3 provides the efficiency diagram of the parallel algorithm for the problem instance. We may conclude that, once there are 16 processors in use, further increasing the number of processors does not result in a significant impact on the solution time. This is due to the rising cost of information exchange between parallel processors.

Figure 1. Diagram of solution time/number of processors.
For a problem instance with 100 circles (40 circles with radius 10, 30 with radius 20, 20 with radius 30 and 10 with radius 40) the sequential algorithm results in $r_{up} = 257.35311$. For the same instance, the parallel algorithm with eight processors and 100,000 starting points gives an improved value $r_{up} = 257.1951$ in 64,649 s (approximately 17 h 58 min). Sequential and parallel solutions are presented in Figures 4 and 5, respectively, while Tables 2 and 3 provide corresponding center coordinates. As can be seen from Figures 4 and 5, the layout for the circles differs significantly.

**Table 2.** Coordinates of the placement of 100 circles for the sequential solution, $r_{up} = 257.35311$.

| i  | $x_{i}^{up}$ | $y_{i}^{up}$ | i  | $x_{i}^{up}$ | $y_{i}^{up}$ | i  | $x_{i}^{up}$ | $y_{i}^{up}$ |
|----|--------------|--------------|----|--------------|--------------|----|--------------|--------------|
| 1  | 139.4106     | −148.7203    | 35 | −192.0113    | 152.7411     | 69 | −57.0781     | 139.3253     |
| 2  | 165.9300     | −180.7356    | 36 | −20.2820     | 57.6283      | 70 | 65.7095      | −121.7164    |
| 3  | −10.4564     | 139.3668     | 37 | −33.4277     | −76.2563     | 71 | −60.8956     | 205.8394     |
| 4  | 90.6188      | −228.0052    | 38 | −25.6351     | −26.4029     | 72 | −66.4087     | −215.3460    |
$$\begin{array}{cccccccc}
i & x_i^{up} & y_i^{up} & i & x_i^{up} & y_i^{up} & i & x_i^{up} & y_i^{up} \\
5 & -85.8170 & -120.5509 & 39 & 153.5649 & 88.7727 & 73 & 95.7131 & 204.0172 \\
6 & 73.5905 & 127.7710 & 40 & 14.8656 & 116.2411 & 74 & -174.8863 & 52.7062 \\
7 & -157.6668 & 127.7678 & 41 & 5.8499 & -16.7581 & 75 & 31.2989 & -218.4935 \\
8 & -177.5573 & 169.3267 & 42 & 65.9473 & 43.6220 & 76 & -23.6507 & 99.4930 \\
9 & -70.5784 & -169.3974 & 43 & -25.8560 & 167.4177 & 77 & -224.7904 & 15.9151 \\
10 & 95.7456 & 55.2866 & 44 & 50.5852 & 229.8526 & 78 & -81.3118 & 12.9985 \\
11 & -109.3281 & 44.9178 & 45 & 175.1576 & 157.1970 & 79 & -170.0251 & -13.1497 \\
12 & 150.5197 & 110.5609 & 46 & 125.6546 & 43.9088 & 80 & 219.2165 & 52.2319 \\
13 & -74.5636 & 166.1256 & 47 & 178.0610 & -69.8027 & 81 & -16.3268 & -178.7973 \\
14 & -171.4504 & 144.9147 & 48 & -29.8828 & 5.3140 & 82 & -196.1507 & 110.9455 \\
15 & -15.5382 & -136.8047 & 49 & 137.0313 & 139.5792 & 83 & -62.7066 & -46.1440 \\
16 & 171.8182 & 16.2934 & 50 & -18.1262 & -234.6541 & 84 & 14.7585 & 34.4732 \\
17 & -39.7282 & 242.1133 & 51 & -53.7898 & 57.1185 & 85 & -1.2195 & 225.3498 \\
18 & 141.1580 & -200.6803 & 52 & -47.3537 & -137.2039 & 86 & 111.8985 & 94.0562 \\
19 & -158.9000 & 91.5448 & 53 & 187.9781 & -110.6151 & 87 & 49.9061 & 93.0860 \\
20 & 201.8791 & -139.4381 & 54 & 34.1020 & -166.1113 & 88 & -194.0441 & -70.3082 \\
21 & 37.9330 & -105.8277 & 55 & 165.9846 & 59.2811 & 89 & 192.4508 & 108.1568 \\
22 & -28.6723 & 37.2911 & 56 & 96.1916 & 152.0312 & 90 & -183.1233 & -131.3388 \\
23 & -235.9375 & -67.3175 & 57 & 15.3707 & -128.5195 & 91 & 95.2733 & -176.2139 \\
24 & -132.9352 & 54.7309 & 58 & 171.4069 & -149.2078 & 92 & -122.5708 & -170.2925 \\
25 & 93.6021 & 27.5218 & 59 & -62.9718 & -98.1434 & 93 & 214.4583 & -19.6107 \\
26 & 146.8699 & -169.7488 & 60 & 218.3642 & -81.6204 & 94 & 132.4803 & -17.7144 \\
27 & 11.3898 & 76.3379 & 61 & -225.3420 & 67.9122 & 95 & -101.7842 & 96.3677 \\
28 & 34.5368 & -2.5784 & 62 & 73.8224 & 2.3670 & 96 & 53.6401 & -56.2562 \\
29 & 154.2352 & 29.5161 & 63 & 144.4278 & 185.8271 & 97 & -123.5010 & -84.7919 \\
30 & -224.2984 & -99.4403 & 64 & -8.4552 & -56.2468 & 98 & -125.8419 & 174.7592 \\
31 & 69.8306 & -235.2059 & 65 & -232.6625 & -35.4856 & 99 & 123.4839 & -99.2194 \\
32 & -174.5260 & -172.4495 & 66 & -132.3921 & 22.7355 & 100 & 36.0347 & 163.7371 \\
33 & 5.5154 & -85.0361 & 67 & -118.9449 & -22.8867 & & & \\
34 & -79.2399 & -141.5447 & 68 & -19.4571 & -105.0456 & & & \\
\end{array}$$

Table 3. Coordinates of the placement of 100 circles for the parallel solution, \( r_{up} = 257.195. \)
| \(i\) | \(x_{ij}^{up}\) | \(y_{ij}^{up}\) | \(i\) | \(x_{ij}^{up}\) | \(y_{ij}^{up}\) | \(i\) | \(x_{ij}^{up}\) | \(y_{ij}^{up}\) |
|---|---|---|---|---|---|---|---|---|
| 16 | -167.0885 | -73.2191 | 50 | 86.9860 | 137.7383 | 84 | -124.5224 | -47.9457 |
| 17 | -190.4386 | 2.9106 | 51 | -23.2266 | -49.0803 | 85 | -68.3349 | 147.8602 |
| 18 | 115.1268 | -131.2206 | 52 | -11.5071 | -103.2896 | 86 | -209.5402 | -82.4969 |
| 19 | -109.4713 | 156.3334 | 53 | -194.4786 | -132.2679 | 87 | -189.9519 | 120.9591 |
| 20 | 101.1534 | -78.3999 | 54 | 218.3378 | 87.4377 | 88 | 177.8639 | -53.9579 |
| 21 | 20.2577 | 108.4091 | 55 | -42.5723 | 75.0849 | 89 | 133.2627 | 55.4251 |
| 22 | -54.4610 | -35.0199 | 56 | 31.4821 | -93.8401 | 90 | 51.6126 | -141.7886 |
| 23 | 176.9704 | 39.4416 | 57 | 69.5126 | -73.6185 | 91 | 74.9291 | 12.9096 |
| 24 | 14.9748 | -121.2538 | 58 | 51.7187 | 114.9257 | 92 | -140.5877 | -162.9234 |
| 25 | 130.6056 | 13.5093 | 59 | 230.4083 | 47.2095 | 93 | 55.7506 | 207.8480 |
| 26 | 198.7563 | 42.5033 | 60 | -157.3516 | 17.1183 | 94 | 171.4856 | 128.0440 |
| 27 | 96.7626 | -56.8425 | 61 | -97.7063 | -213.9397 | 95 | 173.2629 | -127.6280 |
| 28 | -110.8739 | 134.3782 | 62 | 133.0906 | -80.4032 | 96 | -26.5284 | 210.9667 |
| 29 | 161.2199 | 24.0819 | 63 | 92.0306 | -109.0719 | 97 | 112.9423 | -183.1747 |
| 30 | 244.4979 | 18.4780 | 64 | -138.0666 | 117.5075 | 98 | -208.9399 | 51.5080 |
| 31 | 55.0646 | 146.7503 | 65 | -48.7620 | -122.6821 | 99 | 36.2227 | -212.1246 |
| 32 | 0.9221 | -73.8021 | 66 | 218.9989 | -85.7685 | 100 | -6.6694 | 21.0142 |
| 33 | -97.0099 | 73.1228 | 67 | -162.6154 | -104.9049 |
| 34 | -173.1583 | -10.7053 | 68 | -117.5900 | 3.5901 |

Figure 4. Placement of 100 circles for the sequential solution, \(r_{up} = 257.35311\).
2.3. Solving DCBP Using IPOPT

Several instances of the DCBP problem were solved by the NLP solver IPOPT [46] available at https://projects.coin-or.org/Ipopt (accessed on 4 April 2022). A computer with an AMD Athlon 64 X2 5200+ was used for numerical experiments with the decomposition algorithm [31].

Three problem instances were considered, having the following circle distributions characterized by the indicator (the number of equal circles): (their radius), as follows: (a) total 50 circles (20:10, 15:20, 10:30 and 5:40); (b) total 150 circles (50:10, 40:20, 30:30, 20:40 and 10:50); (c) total 300 circles, two collections of the 150 circles specified in (b). It was assumed that $w_i = r_i$ for $i = 1, 2, \ldots, n$.

Figure 6 shows arrangements of circles corresponding to the local minima of DCBP for cases (a), (b) and (c).
Figure 6. Locally optimal layouts of circles in DCBP: (a) \( n = 50, r^* = 182.6996, \) CPU 1355.85 s for 100 runs; (b) \( n = 150, r^* = 368.4018, \) CPU 45910.3 s for 50 runs; (c) \( n = 300, r^* = 520.5562, \) CPU 24185.24 s for 50 runs.

3. Sparse Balanced Packing Problem with Balancing Conditions

In this section, a sparse circular packing problem is considered under balancing conditions. The objective is to place the circles in a fixed circular container maximizing the smallest distance between the circles, as well as the distance between the circles and the boundary of the container.

Denote the Euclidean distance between two circles by \( d_{ij} \) for \( i > j = 1, 2, \ldots, n \) and the distance between a circle and the boundary of the circular container by \( d_i, i = 1, 2, \ldots, n. \) Let \( d = \min \{ d_{ij} i > j = 1, 2, \ldots, n, d_i, i = 1, 2, \ldots, n \} \) be the minimum among all these distances.
A sparse circular balanced packing (SCBP) problem is as follows. Pack the circles $S_i$, $i = 1, 2, \ldots, n$, into a fixed circular container $S$, maximizing $d$, subject to a limited deviation between the origin and the gravitation center of all circles $S_i$, $i = 1, 2, \ldots, n$. The maximal allowed deviation is denoted by $\varepsilon$.

The SCBP problem is stated as follows:

$$d^* = \max_{d > 0, x, y} d$$

subject to

$$(x_i - x_j)^2 + (y_i - y_j)^2 \geq (r_i + r_j + d)^2, \quad 1 \leq i < j \leq n,$$

$$x_i^2 + y_i^2 \leq (r_i - d)^2, \quad i = 1, \ldots, n,$$

$$-\varepsilon \leq \sum_{i=1}^{n} w_i x_i \leq \varepsilon, \quad -\varepsilon \leq \sum_{i=1}^{n} w_i y_i \leq \varepsilon.$$

The constraints in (12) and (13) assure that the corresponding distances are at least $d$. The constraint in (14) guarantees that the component-wise difference between the center of gravity and the origin (the center of the container) is at most $\varepsilon$.

The problem in (11)–(14) has $n(n-1)/2$ quadratic nonconvex constraints (12), $n$ quadratic non-convex constraints (13) and $4n$ linear balanced constraints (14).

The following technique is applied for the problem in (11)–(14):

Stage 1. Generate several feasible starting points by homothetic transformations of the circles [21].

Stage 2. Obtain a local minimum for (11)–(14), starting from the points obtained at Stage 1 and using the optimization procedure proposed in [47]. The procedure substitutes solving the problem in (11)–(14) with $O(n^2)$ nonlinear constraints by analyses of several optimization subproblems with $O(n)$ constraints.

Stage 3. Consider the best local solution from Stage 2 as a solution to the original problem in (11)–(14).

For computational experiments, a computer with an AMD Athlon 64 X2 5200+ was used.

Three problem instances were considered: (a) total 50 circles (20:10, 15:20, 10:30 and 5:40); (b) total 100 circles (40:10, 30:20, 20:30 and 10:40); (c) total 150 circles (50:10, 40:20, 30:30, 20:40 and 10:50). Note that here, instances (a) and (c) coincide with instances (a) and (b) from Section 2. For all problem instances, $\varepsilon = 0$ and $w_i = r_i$ for $i = 1, 2, \ldots, n$. Moreover, the radii of the containers in instances (a) and (b) were chosen as follows: $R_a = 182.6996$, $R_b = 257.3531$, $R_c = 368.4018$, such that $R_a$ and $R_c$ coincide with corresponding radii obtained in Section 2 for the dense packing problem. The radius of container $S$ was found as a solution to the dense packing problem considered in the previous section.

Figure 7 shows arrangements of circles corresponding to the local minima of SCBP for cases (a), (b) and (c).
4. Conclusions

Two circular packing problems are considered in this work: dense and sparse. In the first problem, the objective is to find the smallest possible circular container under distance and balancing conditions for the circular objects. In the second problem, the circular container is fixed, and the objective is to locate the circles as distant from one another as possible, as well as from the boundary of the container.

A mathematical model and sequential/parallel algorithms are presented for the dense packing problem. Parallel algorithms use computing technology with distributed memory (clusters). Solution algorithms apply the multistart strategy together with non-smooth minimization methods to search for the local minimum. The results of computational experiments for parallel algorithms on cluster SKIT-4 from the V.M. Glushkov Institute of Cybernetics of the National Academy of Sciences of Ukraine are presented. It is demonstrated that using the parallel approach can significantly reduce the solution time. The parallel solution allows finding good local minima for problem instances with several
hundred circles in a reasonable time. The parallel solution also results in an improved value for the container’s radius compared with the sequential technique.

The mathematical formulation for the sparse circular packing problem was presented. The corresponding nonlinear optimization problem was solved by the open-source local solver IPOPT [46], combined with the decomposition algorithm in [47] to cope with large dimensions. Numerical results are presented for the arrangement of circular containers obtained by dense packing. The corresponding solutions provide a visually more uniform distribution of the circles compared with dense packing. Obtaining a sparse layout is important in many practical applications, e.g., in thermal deburring [39], where it is used to ensure the uniformity of power and thermal effects in a deburring chamber (container).

Both formulations considered in this work result in large-scale optimization problems. To use the specific structure of the constraints, special-purpose decomposition/aggregation techniques [48,49] can be used to either relax the binding constraints or form an approximate aggregated problem of lower dimension.

An interesting direction for future research is using parallel algorithms for irregular packing problems with distance and balancing conditions. Some results in this direction are on the way [50]. Parallel techniques used in this work demonstrate their efficiency for traditional optimization approaches. We may expect that combining evolution algorithms [33] with parallel frameworks for packing problems may provide more computational savings.

**Author Contributions:** Conceptualization, T.R. and O.P.; methodology, T.R., O.P., I.L. and J.A.M.-S.; software, O.P. and O.L.; validation, I.L. and J.A.M.-S.; formal analysis, T.R.; investigation, P.S. and P.V.; resources, T.R.; data curation, O.P. and O.L.; writing—original draft preparation, T.R.; writing—review and editing, J.A.M.-S. and I.L.; visualization, O.P.; supervision, T.R.; project administration, T.R. and P.V. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Acknowledgments:** The research was supported by the Volkswagen Foundation (grant #Az97775, P. Stetsyuk and T. Romanova, grant #9C086, T. Romanova) and the National Research Foundation of Ukraine (grant # 02.2020/167, T. Romanova).

**Conflicts of Interest:** The authors declare no conflict of interest.

**Appendix A. Shor’s r(α)-algorithm**

Here, we give a short description of the r(α)-algorithm. This algorithm belongs to the family of subgradient methods for minimizing non-smooth convex functions known as Shor’s r-algorithms. These algorithms use the steepest possible descent combined with space transformation and ensure monotonic (or almost monotonic) objective function decrees. A space transformation in the direction of the difference of two subsequent subgradients is used in r-algorithms; this improves the behavior of ravine functions in the transformed space of the variables.

Let \( x \) be an \( n \)-vector of variables and \( f(x) \) be a convex function. Assuming \( \alpha > 1 \), using the r(α)-algorithm to minimize \( f(x) \) is an iterative technique to obtain vectors \( \{x_k\}_{k=0}^{\infty} \) and matrices \( \{B_k\}_{k=0}^{\infty} \) such that:

\[
x_{k+1} = x_k - h_k B_k \xi_k, \quad B_{k+1} = B_k R_\beta (\tau_k), \quad k = 0, 1, 2, ..., \tag{A1}
\]

where

\[
\xi_k = \frac{B^T_k g_f(x_k)}{\left\| B^T_k g_f(x_k) \right\|}, \quad h_k = \arg \min_{h \geq 0} f(x_k - h B_k \xi_k), \tag{A2}
\]

\[
\tau_k = \frac{B^T_k r_k}{\left\| B^T_k r_k \right\|}, \quad r_k = g_f(x_{k+1}) - g_f(x_k), \quad \beta = \frac{1}{\alpha} < 1. \tag{A3}
\]
In (A1)–(A3), \(x_0\) is an arbitrary starting point, and \(B_0 = I_n\) is an identity \(n \times n\) matrix (\(B_0\) also can be defined as a diagonal matrix \(D_n\) with positive diagonal elements). The step-size multiplier \(h_k\) is obtained by minimizing \(f(x)\) in the direction of the subgradient in the transformed space; the coefficient \(\alpha\) is used for space dilation, and the operator \(R_{\beta}(\tau) = I_n + (\beta - 1) \cdot \tau \cdot \tau^T\) is used for “pressing” subgradients in the normalized direction \(\tau\) with a coefficient \(\beta = \frac{1}{\alpha} < 1\). Vectors \(g_j(x_k)\) and \(g_j(x_{k+1})\) denote subgradients of \(f(x)\) at \(x_k\) and \(x_{k+1}\). The process in (14)–(16) stops if \(g_j(x_k) = 0\) and \(x_k\) is a minimum point of \(f(x)\).

This algorithm is implemented as an Octave program ralgb5. This program can find an approximate value of \(x^*_r\), which is a minimizer of the function \(f(x)\). Its name indicates that the recalculation of the matrix \(B\) in algorithms (A1)–(A3) needs the \(5n^2\) arithmetical operation. The program ralgb5 can find more precise approximations \(x^*_r\) than the existing program ralgb4, economizing \(n^2\) arithmetical operations per iteration. Control parameters are the same in both programs.

Adaptive step regulation is related to approximately minimizing the objective function in the subgradient direction and uses \(h_0, q_1, n_h,\) and \(q_2\) as parameters. Here, \(h_0\) is an initial step-size (this value is updated after the first iteration), \(q_1\) is a coefficient of the step reduction (\(q_1 \leq 1\)) if the condition of stopping descent is satisfied in one step and \(q_2\) is a coefficient of the step increasing (\(q_2 \geq 1\)). The integer \(n_h\) denotes the number of steps in a one-dimensional descent (\(n_h > 1\)). After \(n_h\) steps, the step size is \(q_2\) times greater.

The choice of the coefficient of space transformation and the parameters of the adaptive step-size regulation aims to improve the accuracy of minimizing the function along a direction, with the number of steps along a direction being moderate.

Stopping rules for the \(r(\alpha)\)-algorithm are defined by the parameters \(e_x\) and \(e_g\); iterations are terminated at \(x_{k+1}\) if \(\|x_{k+1} - x_k\| \leq e_x\) (criterion in the space of variables) or if \(\|g_j(x_{k+1})\| \leq e_g\) (criterion in the objective space, used for smooth functions). For values of \(f(x)\) not bounded below, or for too-small values of the initial step \(h_0\), the algorithm may terminate in an abnormal manner. In the latter case, the initial step-size must be enlarged.

The following parameter values are recommended for minimizing non-smooth functions: \(\alpha = 2 \div 3, h_0 = 1.0, q_1 = 1.0, q_2 = 1.1 \div 1.2, n_h = 2 \div 3\). If a priori estimation of the distance between a starting point \(x_0\) and a minimum point \(x^*\) is available, then a reasonable choice for the initial step \(h_0\) is approximately \(\|x_0 - x^*\|\).

Basically, the same parameters can be used for minimizing smooth functions. However, \(q_1 = 0.8 \div 0.95\) is recommended. The reason is that the additional size decomposition provides a more accurate directional minimum of the function; for smooth functions, this provides a better convergence rate. For these parameters, the number of descents is usually less than two, and after \(n\) steps, the accuracy in terms of the objective is three to five times better. Choosing \(e_x, e_g \sim 10^{-6} \div 10^{-5}\) for the minimization of a convex function gives good approximations of the optimal objective.

According to numerous tests and applied calculations, for non-smooth functions, we usually have \(\frac{f(x^*) - f(x^0)}{f(x^0)} \sim 10^{-5} \div 10^{-5} (\sim 10^{-12} \div 10^{-10}\) for smooth functions).

References

1. Bowers, P.L.; Stephenson, K. Uniformizing Dessins and BelyiMaps via Circle Packing; American Mathematical Soc.: Providence, RI, USA, 2004; Volume 170.
2. Stephenson, K. Introduction to Circle Packing: The Theory of Discrete Analytic Functions; Cambridge University Press: Cambridge, UK, 2005.
3. Bowers, P.L. Introduction to Circle Packing: A Review. 2005. Available online: https://www.math.fsu.edu/~aluffi/archive/paper356.pdf (accessed on 27 March 2022).
4. Hifi, M.; M’Hallah, R. A Literature Review on Circle and Sphere Packing Problems: Models and Methodologies. Adv. Oper. Res. 2009, 2009, 1–22. https://doi.org/10.1155/2009/150624.
5. Castillo, I.; Kamps, F.J.; Pintér, J.D. Solving circle packing problems by global optimization: Numerical results and industrial applications. Eur. J. Oper. Res. 2008, 191, 786–802. https://doi.org/10.1016/j.ejor.2007.01.054.
6. Yagiura, M.; Umetani, S.; Imahori, S.; Hu, Y. Cutting and Packing Problems: From the Perspective of Combinatorial Optimization; Springer: Berlin/Heidelberg, Germany, 2022.
7. Zeng, Z.; Yu, X.; He, K.; Huang, W.; Fu, Z. Iterated tabu search and variable neighborhood descent for packing un-equal circles into a circular container. *Eur. J. Oper. Res.* 2016, 250, 615–627.
8. He, K.; Ye, H.; Wang, Z.; Liu, J. An efficient quasi-physical quasi-human algorithm for packing equal circles in a circular container. *Comput. Oper. Res.* 2018, 92, 26–36. https://doi.org/10.1016/j.cor.2017.12.002.
9. Astarkov, S. On regular covering/packing of the Euclidean plane with circles. In *Proceedings of the International Conference on Geometric Analysis in Honor of the 90th Anniversary of Academician Yu. G. Reshetynyak*, Novosibirsk, Russia, 22–28 September 2019; pp. 13–15.
10. Huang, W.Q.; Li, Y.; Akeb, H.; Li, C.M. Greedy algorithms for packing unequal circles into a rectangular container. *J. Oper. Res. Soc.* 2005, 56, 539–548. https://doi.org/10.1057/palgrave.jors.2601836.
11. Birgin, E.; Sobral, F. Minimizing the object dimensions in circle and sphere packing problems. *Comput. Oper. Res.* 2008, 35, 2357–2375. https://doi.org/10.1016/j.cor.2006.11.002.
12. Birgin, E.G.; Gentil, J.M. New and improved results for packing identical unitary radius circles within triangles, rectangles and strips. *Comput. Oper. Res.* 2010, 37, 1318–1327.
13. Akeb, H.; Hifi, M. Algorithms for the circular two-dimensional open dimension problem. *Int. Trans. Oper. Res.* 2008, 15, 685–704. https://doi.org/10.1111/j.1475-9995.2008.00655.x.
14. Akeb, H.; Hifi, M.; Negre, S. An augmented beam search-based algorithm for the circular open dimension problem. *Comput. Ind. Eng.* 2011, 61, 373–381.
15. Stoyan, Y.; Yaskov, G. Packing equal circles into a circle with circular prohibited areas. *Int. J. Comput. Math.* 2012, 89, 1355–1369. https://doi.org/10.1080/00207160.2012.685468.
16. Zhuang, X.; Yan, L.; Chen, L. Packing equal circles in a damaged square. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, Killarney, Ireland, 12–17 July 2015, pp. 1–6.
17. Kazakov, A.L.; Lemptert, A.A.E.; Nguyen, H.L. An algorithm of packing congruent circles in a multiply connect-ed set with non-euclidean metrics. *Numer. Methods Program.* 2016, 17, 177–188.
18. López, C.O.; Beasley, J.E. Packing a fixed number of identical circles in a circular container with circular prohibited areas. *Optim. Lett.* 2018, 13, 1449–1468. https://doi.org/10.1007/s11590-018-1351-x.
19. He, K.; Mo, D.; Ye, T.; Huang, W. A coarse-to-fine quasi-physical optimization method for solving the circle pack-ing problem with equilibrium constraints. *Comput. Ind. Eng.* 2013, 66, 1049–1060.
20. Kovalenko, A.; Romanova, T.; Stetsyyuk, P. Balance packing problem for 3D-objects: Mathematical model and solution methods. *Cybern. Syst. Anal.* 2015, 51, 556–565.
21. Stetsyyuk, P.; Romanova, T.; Scheithauer, G. On the global minimum in a balanced circular packing problem. *Optim. Lett.* 2016, 10, 347–360.
22. Fasano, G.; Pinter, J.D. *Modeling and Optimization in Space Engineering: State of the Art and New Challenges*; Springer: Berlin/Heidelberg, Germany, 2019.
23. Wang, Y.; Wang, Y.; Sun, J.; Huang, C.; Zhang, X. A stimulus–response-based allocation method for the circle packing problem with equilibrium constraints. *Phys. A Stat. Mech. Its Appl.* 2019, 522, 232–247. https://doi.org/10.1016/j.physa.2019.02.007.
24. Litvinchev, I.; Infante, L.; Ozuna, L. Packing circular-like objects in a rectangular container. *J. Comput. Syst. Sci. Int.* 2015, 54, 259–267. https://doi.org/10.1134/s1064226615020070.
25. Kazakov, A.L.; Lemptert, A.A.; Nguyen, H.L. The Problem of the Optimal Packing of the Equal Circles for Special Non-Euclidean Metric. In *International Conference on Analysis of Images, Social Networks and Texts*; Springer: Cham, Switzerland, 2017; pp. 58–68. https://doi.org/10.1007/978-3-319-52920-2_6.
26. Litvinchev, I.; Ozuna, E. Approximate Packing Circles in a Rectangular Container: Valid Inequalities and Nesting. *J. Appl. Res. Technol.* 2014, 12, 716–723. https://doi.org/10.1016/s1665-6423(14)70088-4.
27. Pedroso, J.P.; Cunha, S.; Tavares, J.N. Recursive circle packing problems. *Int. Trans. Oper. Res.* 2016, 23, 355–368.
28. Gleixner, A.; Maher, S.J.; Müller, B.; Pedroso, J.P. Price-and-verify: A new algorithm for recursive circle packing using Dantzig–Wolfe decomposition. *Ann. Oper. Res.* 2018, 284, 527–555. https://doi.org/10.1007/s10479-018-3115-5.
29. Ekanayake, D.B.; Ranapatidewage, M.M.; LaFountain, D.J. Optimal packings for filled rings of circles. *Appl. Math. 2020*, 65, 1–22. https://doi.org/10.21136/am.2020.0244-19.
30. Scholz, M.; Dietmeier, M. Packing and stacking rings into rectangular bins. *Procedia Comput. Sci.* 2022, 200, 768–777. https://doi.org/10.1016/j.procs.2022.01.275.
31. Miyazawa, F.K.; Wakabayashi, Y. Techniques and results on approximation algorithms for packing problems. *São Paulo J. Math. Sci.* 2022, 16, 585–615. https://doi.org/10.1007/s40863-022-00301-3.
32. Galiev, S.I.; Lisafina, M.S. Linear models for the approximate solution of the problem of packing equal circles into a given domain. *Eur. J. Oper. Res.* 2013, 230, 505–514. https://doi.org/10.1016/j.ejor.2013.04.050.
33. Torres-Escobar, R.; Marmolejo-Saucedo, J.A.; Litvinchev, I. Binary monkey algorithm for approximate packing non-congruent circles in a rectangular container. *Wirel. Netw. 2020*, 26, 4743–4752.
34. Ryu, J.; Lee, M.; Kim, D.; Kallrath, J.; Sugihara, K.; Kim, D.-S. VOROPACK-D: Real-time disk packing algorithm using Voronoi diagram. *Appl. Math. Comput.* 2020, 375, 125076. https://doi.org/10.1016/j.amc.2020.125076.
35. He, K.; Tole, K.; Ni, F.; Yuan, Y.; Liao, L. Adaptive large neighborhood search for solving the circle bin packing problem. *Comput. Oper. Res.* 2022, 105140. https://doi.org/10.1016/j.cor.2020.105140.

36. Yuan, Y.; Tole, K.; Ni, F.; He, K.; Xiong, Z.; Liu, J. Adaptive simulated annealing with greedy search for the circle bin packing problem. *Comput. Oper. Res.* 2022, 105826. https://doi.org/10.1016/j.cor.2022.105826.

37. E. Specht. 2018. Available from: http://packomania.com (accessed on 4 April 2022).

38. Romanova, T.; Pankratov, A.; Litvinchev, I.; Plankovskyy, S.; Tsegelnyk, Y.; Shypul, O. Sparsest packing of two-dimensional objects. *Int. J. Prod. Res.* 2020, 59, 3900–3915. https://doi.org/10.1080/00207543.2020.1755471.

39. Romanova, T.; Stoyan, Y.; Pankratov, A.; Litvinchev, I.; Avramov, K.; Chernobryvko, M.; Yanchevskyi, I.; Mozgova, I.; Bennell, J. Optimal layout of ellipses and its application for additive manufacturing. *Int. J. Prod. Res.* 2019, 59, 560–575. https://doi.org/10.1080/00207543.2019.1697836.

40. Shor, N.Z. *Nondifferentiable Optimization and Polynomial Problems*; Kluwer Academic: Boston, MA, USA; Dordrecht, The Netherland; London, UK, 1998; p. 412. https://doi.org/10.1007/978-1-4757-6015-6.

41. Stetsyuk, P.I. Shor’s r-Algorithms: Theory and practice. In Optimization Methods and Applications: In Honor of the 80th Birthday of Ivan, V. Sergienko. Butenko, S., Pardalos, P.M., Shylo, V. (Eds). Springer, pp. 495-520, 2017.

42. Stetsyuk, P.I. Theory and Software Implementations of Shor’s r-Algorithms*. *Cybern. Syst. Anal.* 2017, 53, 692–703. https://doi.org/10.1007/s10559-017-9971-1.

43. Shor, N.Z.; Zhurbenko, N.G.; Likhovid, A.P.; Stetsyuk, P.I. Algorithms of Nondifferentiable Optimization: Development and Application. *Cybern. Syst. Anal.* 2003, 39, 537–548. https://doi.org/10.1023/b:casas.000003503.25710.84.

44. Octave [Free Access Electronic Resource]. Available online: http://www.octave.org (accessed on 14 March 2022).

45. Cluster SKIT. Available online: https://icybcluster.org.ua/ (accessed on 4 April 2022).

46. Wächter, A.; Biegler, L.T. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.* 2005, 106, 25–57. https://doi.org/10.1007/s10107-004-0559-y.

47. Romanova, T.; Stoyan, Y.; Pankratov, A.; Litvinchev, I.; Marmolejo, J.A. Decomposition Algorithm for Irregular Placement Problems. In *Advances in Intelligent Systems and Computing*; Intelligent Computing and Optimization ICO 2019; Vasant, P., Zelinka, I., Weber, G.W., Eds.; 2020; Volume 1072, pp. 214–221.

48. Litvinchev, I.S.; Rangel, S. Localization of the optimal solution and a posteriori bounds for aggregation. *Comput. Oper. Res.* 1999, 26, 967–988. https://doi.org/10.1016/s0305-0548(99)00027-1.

49. Litvinchev, I. Decomposition-aggregation method for convex programming problems. *Optimization* 1991, 22, 47–56. https://doi.org/10.1080/023319393108843642.

50. Romanova, T.E.; Stetsyuk, P.I.; Chugay, A.M.; Shekhovtsov, S.B. Parallel Computing Technologies for Solving Optimization Problems of Geometric Design. *Cybern. Syst. Anal.* 2019, 55, 894–904.