myjournal manuscript No.
(will be inserted by the editor)

A reliable area reduction technique for solving circle packing problems

Mihály Csaba Markót[1], **Tibor Csendes**[2]

[1] Advanced Concepts Team, ESTEC, European Space Agency. Keplerlaan 1, 2201 AZ Noordwijk, The Netherlands.
[2] University of Szeged, Institute of Informatics. H-6701 Szeged, P.O. Box 652, Hungary.

Received: date / Revised version: date

Abstract We are dealing with the optimal, i.e. densest packings of congruent circles into the unit square. In the recent years we have built a numerically reliable, verified method using interval arithmetic computations, which can be regarded as a 'computer-assisted proof'. An efficient algorithm has been published earlier for eliminating large sets of suboptimal points of the equivalent point packing problem. The present paper discusses an interval arithmetic based version of this tool, implemented as an accelerating device of an interval branch–and–bound optimization algorithm. In order to satisfy the rigorous requirements of a computational proof, a detailed algorithmic description and a proof of correctness are provided. This elimination method played a key role in solving the previously open problem instances of packing 28, 29, and 30 circles.

Key words circle packing, interval arithmetic, area reduction, computer-assisted proof

AMS subject classification 52C15, 52C26, 65G30, 90C30, 90C57

1 Introduction

The original circle packing problem is the following: *place a given number n of congruent circles without overlapping into a unit square maximizing the diameter of the circles*. It is easy to see that an equivalent problem is given by *placing a given number n of points into the unit square maximizing the minimal squared distance between the pairs of points* [14]. We call the latter problem setting as *point packing*. An optimal solution of the circle packing problem is determined by an optimal solution of the point packing problem, and vice versa.

Formally, we are looking for all optimal solutions of

$$\text{maximize} \min_{1\le i\neq j\le n} (x_i - x_j)^2 + (y_i - y_j)^2, \tag{1}$$

$$\text{s.t.} \quad 0 \le x_i, y_i \le 1, \qquad i = 1, 2, \ldots, n,$$

where $[0,1]^2$ is the unit square, and we place the ith point at (x_i, y_i).

The objective function of the point packing problem can be written as

$$f_n : \mathbb{R}^{2n} \to \mathbb{R}, \quad f_n(x,y) = \min_{1\le i\neq j\le n} (x_i - x_j)^2 + (y_i - y_j)^2, \tag{2}$$

with $0 \le x_i, y_i \le 1,\ i = 1, 2, \ldots, n$.

Among other results obtained on this field, Locatelli and Raber has published a DC based deterministic algorithm for the solution of circle packing problem [5]. At one hand they could only locate approximate solutions, and the other hand, their technique was not a reliable one: the errors committed e.g. by the roundings were not handled in a verified way. In this sense their results for the $n = 28, 29$, and 30 cases was not the final word. Actually much better approximative results were known earlier. Our approach is aimed to provide proven optimal solutions to these problem instances.

In order to handle the inaccuracies emerging in floating point computations, we apply interval computations [3,10,13]. The set of nonempty, real, compact *interval*s is denoted by $\mathbb{I}$, where each interval $A \in \mathbb{I}$ is given by $A = [\underline{A}, \overline{A}] = \{a \in \mathbb{R} \mid \underline{A} \le a \le \overline{A}\}$. Here $\underline{A} \in \mathbb{R}$ and $\overline{A} \in \mathbb{R}$ are called the lower and upper bounds of A, respectively. The vector $X = (X_1, X_2, \ldots, X_n)$, $X \in \mathbb{I}^n$, and $X_i \in \mathbb{I}$ for $i = 1, 2, \ldots, n$ is called an *n-dimensional box*. The basic arithmetic operations can be extended to intervals in such a way that they fulfill the set theoretical definition $A \circ B := \{a \circ b \mid a \in A,\ b \in B\}$. Similarly, the interval extension of a real-valued, continuous elementary function φ (e.g. $\sin, \mathrm{abs}, \log$) is given by $\Phi(A) := \{\varphi(a) \mid a \in A\}$. In practice, the result of an interval operation can be computed simply by using only the lower and upper bounds of the argument intervals, while the result of an elementary function call can be evaluated by applying monotonicity considerations (and thus, often using the function values attained at the endpoints of the argument interval).

In interval software libraries, the rounding errors of the common floating point computations are controlled by directed *outward rounding* procedures when computing the lower and upper bounds of interval-type results [2,4].

Interval arithmetic is a straightforward tool for providing lower and upper bounds for the range of a real, continuous function over a set of points. (Note, that these *enclosures* usually overestimate the exact range.) This observation led to the development of interval branch–and–bound optimization algorithms [3,9,13]. An important feature of such algorithms is the application of the so-called *accelerating test*s. These tools discard those parts of the search space (typically boxes), for which it is guaranteed that they cannot contain global optimizer points.

In [9] we designed and tested an interval B&B global optimization program based on the commonly used interval packages Toolbox for C–XSC [2] and PROFIL/BIAS [4]. This program has several possibilities for controlling the branching process and involves sophisticated accelerating tools for general use. Nevertheless, when we investigated its suitability for solving circle packing problem instances, it turned out that we also need specific accelerating tools to discard large, non-optimal search regions efficiently. This can be achieved by utilizing the geometric properties of the problem class. In addition, we have learned that further features are needed to carry out the global phase of the optimization process – e.g. to handle the symmetric and equivalent packing configurations (resulting otherwise in an astronomical number of equivalent optimizers). The first notable result of this study was a reliable, validated method [6] available to verify the earlier known global optimizers for $n \leq 27$ with some exceptional problem cases (where the number of subintervals turned out to be prohibiting). This method included an accelerating test for the 'method of active areas' (see below) using *rectangular* approximations of the search areas still of interest, and applied a method called 'tiling' [1,11,12] during the global phase. (In the tiling procedure the unit square is divided into non-overlapping regions. The optimization algorithm can then be run on sets of tile combinations.)

In our second, improved algorithm [7,8], a more advanced method of the active areas was implemented using a *polygon* approximation of the non-discarded regions. Moreover, we proposed a new method for handling the so-called free points (or free circles when packing circles), i.e. points that can be slightly moved while keeping optimality. Such points are indicating a positive measure set of global minimizers. Finally, we introduced an enhanced multiphase tiling method. As a result, we have managed to develop a fully interval arithmetic based computer assisted technique for finding *all optimal point packings* in a square, and we have successfully solved the earlier unsolved problem instances of $n = 28, 29$, and 30. (Figure 1 shows the found and verified solutions. The shaded circles show the above mentioned free circles.)

Instead of enumerating the astronomical number of all possible tile combinations, the above, improved algorithm deals with sets of lower dimensional subproblems (subsets of tile combinations). For each such subproblem we start a branch-and-bound search that is not allowed to complete a big number of steps, but instead, it is carefully checked whether all possibilities of shrinking the regions of uncertainty for the position of global minimizer points have been utilized. The substantial decrease in the size of the set of points to be checked was always caused by the method of active areas. It is why the details of the B&B frame algorithm, such as the stopping criterion were not important: with the exception of the very last, high precision refinement phase (see [8]) we have never made more than 5 B&B iterations; this was enough to exclude subproblems leading to suboptimal solutions.

In [7], the first results for $n = 28$ were reported with a substantial decrease of the uncertainty in circle centers. In [8] a more detailed description

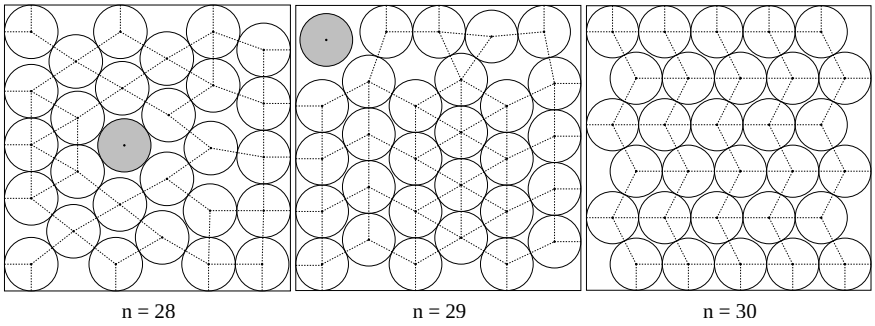


Fig. 1 Verified optimal packing of 28, 29, and 30 circles into the unit square.

of the whole method was given – still without technical details w.r.t. the new method of the active areas. However, since we present a computer-aided *proof*, we must emphasize that a detailed algorithmic description of this tool together with a proof of correctness is essentially required in order to allow checking for correctness and reproducing the results. The present paper is intended to complete this algorithmic description.

The hardware and software environment of the solution procedure was a Pentium IV 1800 MHz computer with 1 GB RAM, using Linux, GNU C/C++, and the interval libraries C–XSC Toolbox and PROFIL/BIAS [2,4]. The bounds and their widths obtained and first published in [8] for the respective optimal values of (2) were

$$F_{28}^* = [\underline{0.2305354936426673}, \underline{0.2305354936426743}],\ w \approx 7 \cdot 10^{-15},$$

$$F_{29}^* = [\underline{0.2268829007442089}, \underline{0.2268829007442240}],\ w \approx 2 \cdot 10^{-14},$$

$$F_{30}^* = [\underline{0.2245029645310881}, \underline{0.2245029645310903}],\ w \approx 2 \cdot 10^{-15}.$$

The total running time was approximately 53, 50, and 21 hours, respectively — in contrast to the anticipated execution time of around ten years required by the earlier best technique. The solution needed the storage of around one million of subintervals. According to the calculation of the volumes of the respective sets the uncertainty in the location of the optimal packing circles has been decreased by as much as 711, 764, and 872 orders of magnitude, respectively. The next problem cases for $n = 31$ to $n = 33$ mean closely a thousand times more tile combinations to be checked, and in this way they do not seem to be solvable by the present method in a short time.

In the sequel we describe the already mentioned interval 'method of active areas' as the most effective interval accelerating tool of our B&B algorithm. The proof of correctness of the algorithms is also given.

2 Method of active areas using polygons

Our accelerating tests are based on a guaranteed lower bound of the maximum of (1), denoted by $\tilde{f}$. This value is often called as *cutoff value*, since it enables us to erase points in the investigated box for which the objective function value is less than $\tilde{f}$. On the other hand, we often need a validated $\tilde{f}_0$ lower bound for the *maximum of the minimal pairwise distance* between the pairs of points, i.e. for the maximum of the square root of the objective function. Assuming the existence of a proper $\tilde{f}$ cutoff value, $\tilde{f}_0$ can be obtained simply by an interval square root evaluation:

$$f_s := \sqrt{\tilde{f}} \in \mathbb{I}, \quad \tilde{f}_0 := \underline{f_s}. \tag{3}$$

The 'method of active areas' or 'active regions' is known from the literature (e.g. [1,11,12]) as a part of non-interval type methods. The key idea of this method is the following: Assume that we have a validated $\tilde{f}_0$ lower bound for the maximum of the minimal pairwise distance between the pairs of points. Considering $(X, Y) \subseteq [0,1]^{2n}$ as a subbox of the search space $[0,1]^{2n}$, from each (X_i, Y_i) (corresponding to a rectangle enclosing the ith point to be packed) one can iteratively delete those points which have a distance smaller than $\tilde{f}_0$ to *all* points of another rectangle $(X_j, Y_j),\ i \neq j$.

First we outline a possible basis algorithm for the method of active areas (see Algorithm 1): Consider a box $(X, Y) \subseteq [0,1]^{2n}$. The ith component of X and Y ($(X_i, Y_i) \subseteq [0,1]^2$) is called the ith *initial active region*, $i = 1, \ldots, n$. During the procedure the active regions R_i of the different components are reduced iteratively, until either one of the active regions becomes empty or a pre-given iteration limit $It_{\max}$ is reached. In the first case the whole box (X, Y) can be erased (Step 5). In the latter case a new box (X', Y') containing the remaining regions will be stored (Steps 7 and 8). The most important part of Algorithm 1 is Step 4 in which we delete some points (forming a so-called *inactive region*) of R_i having a distance smaller than $\tilde{f}_0$ from each point of R_j.

One crucial part of the algorithm is the representation of the intermediate active areas (i.e. the regions R_i). One can easily show that a set of points within a 2-dimensional geometrical object having a distance of at least $\tilde{f}_0$ from all points of another object may be non-convex or non-connected. Nevertheless, a good approximation of the active and inactive point sets is vital to erase as large inactive sets as possible. In [1], the initial active regions were quantized into many rectangular pieces applying splittings both in horizontal and in vertical directions, and the set of eliminated and remaining pieces represented the inactive and active point sets, respectively. In [11], a similar approach was applied but using only splittings in one direction. Until now, the most effective realization is the one of Nurmela and Östergård [12], which approximates the active and inactive regions by polygons. Although in a method working basically with multidimensional intervals the latter solution is more difficult to implement, we found that this extra effort

Algorithm 1 *Method_of_active_areas*

Inputs: – $\tilde{f}_0$: a validated lower bound of the square root of the global maximum of (1),
– $(X, Y) \subseteq [0,1]^{2n}$: the box to be reduced,
– $It_{\max}$: the iteration limit.
Output: – $(X', Y') \subseteq [0,1]^{2n}$: a box containing the remaining areas.

1: **for** $i := 1$ to n **do** $R_i := (X_i, Y_i)$;
2: **for** $k := 1$ to $It_{\max}$ **do**
3: **for all** (i, j), $1 \le i, j \le n$, $i \ne j$ **do**
4: $R'_i := Diminish\_ij((R_i, R_j), \tilde{f}_0)$; {comment: reduce R_i to R'_i}
5: **if** $(R'_i = \emptyset)$ **then return** "(X', Y') is empty";
6: $R_i := R'_i$;
7: **for** $i := 1$ **to** n **do** $(X'_i, Y'_i) := Rectangular\_enclosure(R_i)$;
8: **return** (X', Y');

resulted in an outstanding improvement in the computational efficiency as compared to the method using rectangular approximations. (Notice, that the branching step of the B&B algorithm generates rectangular splittings in a moderate way, thus, it is mixing the advantages of the different approximation techniques.)

The method of [12] is based on the following Lemma and Theorem, and demonstrated in Figure 2.

Lemma 1 [12]: *If a point p is at a distance less than $\tilde{f}_0$ from all the vertices of a polygon R, it is at a distance less than $\tilde{f}_0$ from all points of R.*

Theorem 1 [12]: *Assume that $p_1, \ldots, p_k$ are distinct points on the boundary of a polygon R_i, such that the line segments $\overline{p_l p_{l+1}}$ for $2 \le l \le k-2$ are edges of R_i, and that $\overline{p_1 p_2}$ and $\overline{p_{k-1} p_k}$ lay on the edges of R_i. If the points p_i, $1 \le i \le k$ are at a distance less than $\tilde{f}_0$ from all vertices of R_j $(i \ne j)$, then the points in the polygon formed by $p_1, p_2, \ldots, p_k$ are at a distance less than $\tilde{f}_0$ from all points of R_j.*

Let the polygon R_i be determined by the vertices $b_1, \ldots, b_s$, $s \ge 1$. By Theorem 1, the polygon formed by the convex hull of the points $p_1, \ldots, p_k$ contains only inactive points, i.e. it can partly or fully be eliminated.

Definition 1 *We call a polygon with consecutive vertices $p_1, \ldots, p_k$, $k \ge 3$, and with edges $\overline{p_1 p_2}, \ldots, \overline{p_{k-1} p_k}, \overline{p_k p_1}$ to be a* 'simple' polygon*, if each pair of edges has at most one joint point as the joint endpoint of two consecutive edges.*

Invariance criterion: *during the whole running of the interval implementation of Algorithm 1, each R_i, $i = 1, \ldots, n$ is either a point p_1, or a line segment $\overline{p_1 p_2}$, or a 'simple' polygon with consecutive vertices $(p_1, \ldots, p_k)$, $k \ge 3$.*

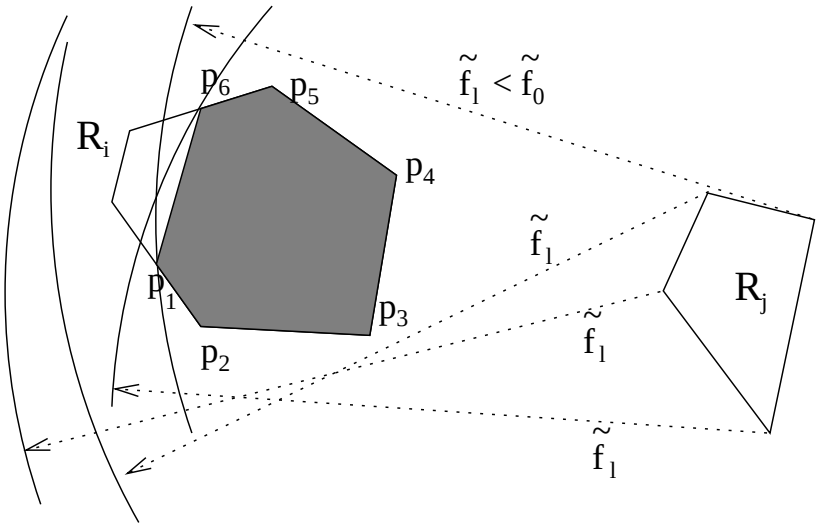


Fig. 2 A basic elimination procedure using polygons ($s = 6, k = 6$) with exact arithmetic. The shaded region of R_i can be considered as the inactive region to be eliminated.

In the sequel we will use the term 'polygon' for figures satisfying the above invariance criterion.

Assuming *exact computations*, one can easily prove that if the polygons $R_i, i = 1, \ldots, n$ are initialized as convex sets (as it is in the current method, see Algorithm 1, Step 1), then they remain convex after each elementary elimination step based on Theorem 1. But with finite precision arithmetic the points p_1 and p_k cannot be evaluated exactly. In the method of Nurmela and Östergård the evaluated points p_1 and p_k are corrected by estimating the possible computation error, while in the present method proper rectangles as the guaranteed enclosures of p_1 and p_k are computed. However, both methods may result in concave, or even self-intersecting R_i polygons. To avoid the difficulty of representing and handling extremely irregular sets, we have to make some restrictions for the shape of the polygons. This was the reason of formulating the above invariance criterion.

The pseudo code of the proposed interval version of an elementary polygon elimination step is given by Algorithm 2. In Algorithm 2 we consider several cases depending on s: $s = 1$ is handled in Steps 4 and 5, while $s = 2$ and $s \geq 3$ are considered in Steps 7 to 10, and Steps 11 to 17, respectively. Note, that $p_0 = p_{k+1}$ may hold in Steps 12 and 13 of Algorithm 2; in this case we construct R_i' without duplicating this point in the result polygon.

We represent polygons commonly as a sequence of consecutive vertices, but we must assume that the coordinates of the vertices are machine numbers. (We start the elimination procedure with such polygons, see Algorithm 1, Step 1.) Each execution of Algorithm 2 results in either an empty set (Step 4), if we can *provide a guarantee* that each vertex of R_i is at a distance less

Algorithm 2 *Diminish_ij* – an interval version

Inputs:
- $R_i = R_i(b_1, b_2, \ldots, b_s)$: the polygon to be reduced,
- $R_j = R_j(a_1, a_2, \ldots, a_t)$: the polygon used for reducing R_i,
- $\tilde{f}_0$: a validated lower bound of the square root of the global maximum of (1).

Output:
- R_i': the remaining polygon of R_i.

1: **for** $l := 1$ to s **do**
2: **if** (it is guaranteed that $d(b_l, a_m) < \tilde{f}_0,\ \forall m = 1, \ldots, t$) **then** mark b_l with a '$-$' flag;
3: **else** mark b_l with a '$+$' flag;
4: **if** (all the b_l have '$-$') **then return** "R_i' is empty";
5: **else if** (all the b_l have '$+$') **then return** $R_i' := R_i$;
6: Find a longest sequence of consecutive vertices b_l with '$-$', denoted by $p_2, \ldots, p_{k-1}$.
7: **if** ($s = 2$) **then** {comment: R_i is a line segment}
8: Denote the node of R_i differing from p_2 by p_0;
9: Find an enclosure $P_1 \in \mathbb{I}^2$ of a point p_1 such that p_1 is on the line segment $\overline{p_0 p_2}$, and $d(p_1, a_m) < \tilde{f}_0,\ \forall m = 1, \ldots, t$.
10: Build R_i' from P_1, p_2;
11: **else** {comment: $s \geq 3$}
12: Denote the preceding node of p_2 in R_i by p_0;
13: Denote the succeeding node of p_{k-1} in R_i by p_{k+1};
14: Find an enclosure $P_1 \in \mathbb{I}^2$ of a point p_1 such that p_1 is on the line segment $\overline{p_0 p_2}$, and $d(p_1, a_m) < \tilde{f}_0,\ \forall m = 1, \ldots, t$.
15: Find an enclosure $P_k \in \mathbb{I}^2$ of a point p_k such that p_k is on the line segment $\overline{p_{k-1} p_{k+1}}$, and $d(p_k, a_m) < \tilde{f}_0,\ \forall m = 1, \ldots, t$.
16: Let $d_1 = p_{k+1}, \ldots, d_{s-k+2} = p_0$ be the consecutive vertices of R_i not chosen in Step 6;
17: Build R_i' from $P_1, P_k, d_1, \ldots, d_{s-k+2}$;
18: **return** R_i';

than $\tilde{f}_0$ from R_j; or a polygon (Step 5 or 18) which contains the polygon that would be obtained assuming exact arithmetic.

Remark 1 Notice, that we need not assume any special properties of the sequence $p_2, \ldots, p_{k-1}$ of Step 6, the only requirement is that it consists of consecutive vertices of R_i marked with '$-$'. For example, if S is such a sequence, then each subsequence S' of consecutive vertices in S can also be considered.

The first problem to be solved when implementing Algorithm 2 is that with the usual floating point arithmetic one cannot decide reliably whether the distance between two points (represented by machine numbers) is less than a given machine number. Instead, we use interval arithmetic as follows:

Algorithm 3 Step 9 and Step 14 of Algorithm 2

Inputs:
- p_0, p_2: consecutive vertices of R_i. p_0 has the flag '+' and p_2 has the flag '−' (by interval computations),
- $R_j = R_j(a_1, \ldots, a_t)$: the reducing polygon,
- $F = \tilde{f}_l^2 \in \mathbb{I}$ such that $\tilde{f}_l < \tilde{f}_0$.
- $(X_i, Y_i) \in \mathbb{I}^2$: the ith initial active region.

Output:
- an inclusion P_1 of an appropriate point p_1.

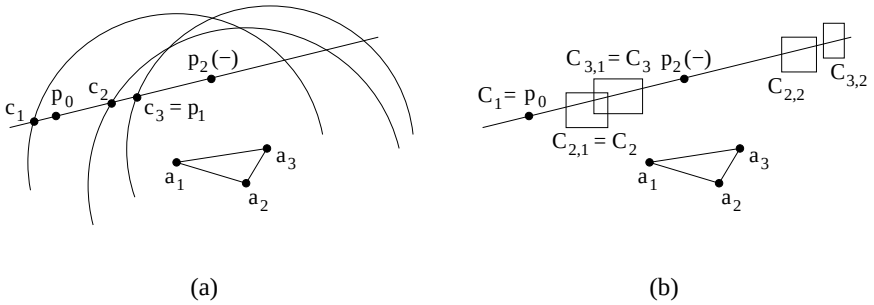
1: **for** $m := 1$ **to** t **do**
2: **if** $(\overline{D}(p_0, a_m) < \underline{F})$ **then** $C_m := p_0$;
3: **else** $C_m := ComputeC(p_2, p_0, a_m, F)$;
4: $ind := \arg\min_{m=1}^{t} \underline{D}(C_m, p_2)$;
5: $P_1 := C_{ind}$;
6: **for** $m := 1$ **to** t **do**
7: **if** $(\overline{D}(C_{ind}, p_2) \geq \underline{D}(C_m, p_2))$ **then** $P_1 := Comp\_Hull(P_1, C_m)$;
8: $P_1 := Intersection(P_1, (X_i, Y_i))$;
9: **return** P_1;

Marking the vertices of R_i by interval computations. Consider an arbitrary vertex $b_l(x_{b_l}, y_{b_l})$ of R_i, and denote the vertices of R_j by $a_1(x_{a_1}, y_{a_1}), \ldots, a_t(x_{a_t}, y_{a_t})$, $t \geq 1$. Moreover, consider a machine number $\tilde{f}_l$ less than $\tilde{f}_0$. Such an $\tilde{f}_l$ can be determined by a direct downward rounding procedure offered by most interval packages. Now compute $D(b_l, a_m) := (x_{b_l} - x_{a_m})^2 + (y_{b_l} - y_{a_m})^2 \in \mathbb{I},\ m = 1, \ldots, t$ by natural interval extension considering each coordinate as point interval and compute $F := \tilde{f}_l^2 \in \mathbb{I}$ also as an interval inclusion. If

$$\overline{D}(b_l, a_m) < \underline{F}, \quad \forall m = 1, \ldots, t,$$

then mark b_l with '−', otherwise mark b_l with '+'. Clearly, b_l receives the flag '−' only in cases when *it is guaranteed* that b_l is at a distance less than $\tilde{f}_l$, and thus less than $\tilde{f}_0$ from all the vertices of R_j. This guarantee has its cost: the resulted set of nodes having the flag '−' may only be a subset of the set of nodes with flag '−' obtained assuming exact computations. However, in accordance with Remark 1 this fact has no effect on the correctness of Algorithm 2.

Computing inclusion rectangles for p_1 and for p_k. The second problem to be solved is to find a reliable alternative of the inaccurate floating-point computation of p_1 and p_k. This is done at Steps 9, 14 and 15 of Algorithm 2. We consider only the case of p_1, a similar process can be introduced for p_k. Clearly, with exact computations, our aim would be *to find a point on the line segment $\overline{p_0 p_2}$ which still can be eliminated, but which is as far from p_2 as possible.* Obviously, p_2 is a suitable choice for P_1 (since it has '−' flag), thus, if problems (due to the overestimation) occur while computing P_1, Step 9 and 14 can still return with $P_1 := p_2$. In the algorithm below we evaluate an inclusion (i.e. a rectangle) $P_1 \in \mathbb{I}^2$ of *an appropriate* p_1 point,



(a) (b)

Fig. 3 Algorithm 3 with exact (left) and interval (right) arithmetic.

thus, we assume that p_1 is on the line determined by p_0 and p_2, and it is at a distance less than $\tilde{f}_0$ from R_j.

A procedure implementing Algorithm 3 with exact computations would work as follows (Figure 3/a): consider the half line H with endpoint p_2 and including p_0p_2. For each a_m compute a point c_m lying on H, where the distance of a_m and c_m is exactly $\tilde{f}_l$ (such c_m points must exist since p_2 has the '$-$' flag). Then find the c_m which is the closest to p_2 and set p_1 to c_m. The case of Figure 3/a results in $p_1 := c_3$. In contrast to the exact computation, the interval algorithm evaluates for each m either

(i) a two dimensional point interval C_m on H which is not farther from p_2 than the exact c_m, or
(ii) a rectangle C_m containing the exact c_m.

Figure 3/b shows the interval version of Algorithm 3, where C_1 is determined by Step 2, while C_2 and C_3 are determined by Step 3.

The aim of the function call $ComputeC(p_2, p_0, a_m, F)$ is to produce an enclosure of the exact c_m when $\overline{D}(p_2, a_m) < \underline{F}$ (this holds since p_2 has a '$-$' flag), and additionally, $\overline{D}(p_0, a_m) \geq \underline{F}$ (thus, when the condition of Step 2 does not hold). In this case the exact c_m must lay on the line segment $\overline{p_0p_2}$, and $p_2 \neq c_m$. Step 3 of Algorithm 3 is always executed at least once, since p_0 has the flag '$+$'. Denoting the coordinates of the corresponding points in the usual way, we have to solve the following system of equations for $c_m(x_{c_m}, y_{c_m})$ in interval way:

$$(y_{c_m} - y_{p_0})(x_{p_2} - x_{p_0}) = (y_{p_2} - y_{p_0})(x_{c_m} - x_{p_0})$$

$$(x_{c_m} - x_{a_m})^2 + (y_{c_m} - y_{a_m})^2 = \tilde{f}_l^2.$$

Here the first equation is equivalent to the statement that c_m lays on the line determined by p_0 and p_2 (when $p_0 \neq p_2$) and the second equality expresses that the distance of c_m to a_m is $\tilde{f}_l$.

This system can be solved in the common way but using interval computations (and handling the possibly different cases arising from the interval valued discriminant). The intermediate computations can be reduced in many steps. Since the basic ideas are clear, the technicalities of solving this

system are not presented in the current paper. Since the interval evaluations may result in significant overestimation (e.g. when one of the result rectangles $C_{m,1}, C_{m,2}$ contains p_2 or when the result boxes are overlapping) we accept the result of the solution procedure only in cases when $C_{m,1}$ and $C_{m,2}$ are disjunct, only one of them contains points from the half line H, and this solution does not contain p_2. (One can easily check the above criteria by comparing the bounds of $C_{m,1}, C_{m,2}, p_0$, and p_2.) In all other cases the algorithm returns $C_m := p_2$ as a 'safety solution'. In Figure 3/b, both for $m = 2$ and for $m = 3$ the result rectangles $C_2 := C_{2,1}$ and $C_3 := C_{3,1}$ can be accepted.

Remark 2 In a very extreme situation the accepted C_m may not contain any points of the line segment $\overline{p_0 p_2}$. In that case C_m is set to p_0 and as a side effect we may obtain $P_1 = p_0$ as a result of Algorithm 3. This means that although p_0 obtained the flag '+' by simple computations during the marking, it would obtain the flag '−' after a more complicated process including ComputeC(). Obviously, this can happen very rarely in practice and did not happen at all in our numerical studies. Consequently, in order to keep our whole method to be as simple as possible, we do not reverse the marking of p_0 in such cases.

Steps 4 to 7 of Algorithm 3 determine P_1 on the basis of the following principle: consider all the possible sets of a number of t points where exactly one point is chosen from each C_m. Then for each set find the element closest to p_2 and give a rectangular enclosure of those closest points. Such an enclosure is given by a componentwise union of several rectangles after we have executed Step 7 a few times. (The function call $Comp\_Hull(P_1, C_m)$ gives the componentwise hull of its 2-dimensional interval arguments.) Steps 4 to 7 of Algorithm 3 implement the procedure above correctly due to the following: Assume that there exists a point combination having its closest point to p_2 within $C_{m_1}, m_1 \neq ind$, where C_{m_1} is not added to P_1. Then $\overline{D}(C_{ind}, p_2) < \underline{D}(C_{m_1}, p_2)$ by Step 7. This means that all the points in C_{ind} are closer to p_2 than any points in C_{m_1}, which contradicts the original assumption. In Figure 3/b ind can be set to 3, and additionally, if $\overline{D}(C_3, p_2) \geq \underline{D}(C_2, p_2)$ and $\overline{D}(C_3, p_2) < \underline{D}(C_1, p_2)$ hold, then P_1 is determined by the componentwise hull of C_2 and C_3.

Step 8 of Algorithm 3 does the rest of the work: since the ith initial active region is the rectangle (X_i, Y_i) (Algorithm 1, Step 1), the result polygon of the exact version of Algorithm 2 is included in (X_i, Y_i). Thus, P_1 can be intersected with this rectangle. Note that the intersection of Step 8 is not empty: we know that $\overline{p_0 p_2} \subseteq (X_i, Y_i)$ must contain a possible result point p_1.

Computing the result polygon R_i'. The only remaining problem to be solved for the interval version of the method of active areas is the determination of R_i', i.e. the implementation of Steps 10 and 17 of Algorithm 2. First we consider the more complicated Step 17:

Algorithm 4 Step 17 of Algorithm 2

Inputs:
- $d_1(=p_{k+1}),\ldots,d_{s-k+2}(=p_0)$: the nonempty set of consecutive vertices not selected in Step 6 of Algorithm 2,
- p_2, p_{k-1}: the first and the last element of the sequence of vertices selected in Step 6 of Algorithm 2,
- P_1, P_k: inclusions of p_1 and p_k, respectively,
- R_i: the polygon to be reduced.

Output:
- R_i': the result polygon.

1: $K := ConvexHull(P_1, P_k)$;
2: **if** $(p_0 \neq p_{k+1})$ **then**
3: **if** $(Separate(\{d_2,\ldots,d_{s-k+1}\},\{P_1,P_k\},\overline{p_0p_{k+1}}))$ **then**
4: **if** $(K' := ConvexHull(p_0, p_{k+1}, K)$ is determined) **then**
5: Let K' be denoted by $K'(p_0, e_1,\ldots,e_u,p_{k+1})$;
6: $R_i' := R_i'(e_1,\ldots,e_u,d_1,\ldots,d_{s-k+2})$;
7: **return** R_i';
8: **if** $(Separate(\{d_2,\ldots,d_{s-k+1}\},\{p_2,p_{k-1}\},\overline{p_0p_{k+1}}))$ **then**
9: $R_i' := R_i'(p_2, p_{k-1}, d_1,\ldots,d_{s-k+2})$;
10: **return** R_i';
11: **else** {comment: $p_0 = p_{k+1}$}
12: **if** $(p_0 \notin K)$ **then**
13: **if** $(R_i' := ConvexHull(p_0, K)$ is determined) **then return** R_i';
14: **return** R_i;

Implementation of Step 17 of Algorithm 2: We have the rectangles P_1 and P_k as inclusions of the points p_1 and p_k, respectively, where p_1 and p_k are two suitable points (but not necessarily the same as for the exact algorithm) of a result polygon of Algorithm 2. For our interval implementation we will define a result polygon having vertices represented by machine numbers. This polygon includes *all the possible result polygons* where p_1 and p_k are chosen arbitrarily from P_1 and P_k, respectively. Our implementation is given by Algorithm 4. The essence of the algorithm is demonstrated by Figure 4.

The key function call of Algorithm 4 is performed in Steps 3 and 8 and is called *Separate*(). It has three parameters: the first one is a set of points, the second one can be a set of points or a pair of rectangles, and the third one is a line L defined by two points. *Separate*() returns a true value only if it is guaranteed that all the elements of the first parameter are located on the one half plane determined by L and all the elements of the second parameter are located on the other half plane. We do not allow touching the line L. The above criterion looks to be a strict restriction, however, it helps us to obtain polygons satisfying the required invariance criterion in an easy way. During the computations *Separate*() returned true in almost all cases. Note that the first parameter set of *Separate*() is allowed to be empty.

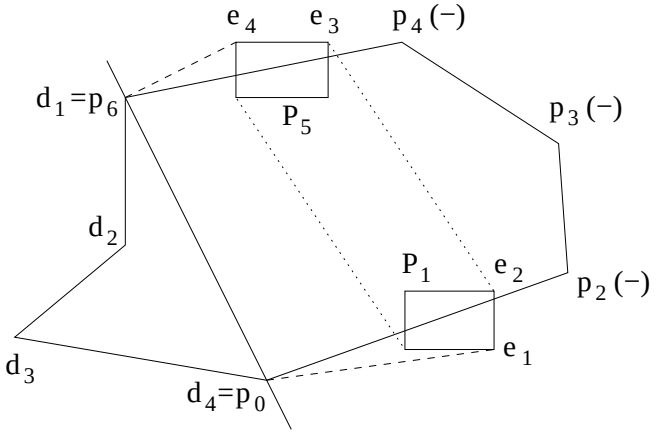


Fig. 4 An example of Algorithm 4 for $s = 7, k = 5, u = 4$. The result polygon is determined by the vertices $e_1, \ldots, e_4, d_1, \ldots, d_4$.

The other important function in Algorithm 4 is $ConvexHull()$, which returns the convex hull polygon of its argument. In general, this could be a difficult problem, especially with finite precision arithmetic. Nevertheless, since both P_1 and P_k are machine representable rectangles (or line segments or points in special cases) with horizontal and vertical bounds, in Step 1 $ConvexHull()$ can easily be determined. In Figure 4 the convex hull of P_1 and P_5 is determined by the two dotted line segments and the appropriate edges of P_1 and P_5.

The evaluation of the convex hull in Step 4 is a slightly harder task. Since it was called a large number of times, we decided to code it for this particular purpose instead of using standard interval tools. If $\{c_v\}$ denotes the set of vertices of K, we have to select that of the $e_1 := c_j$ point, for which the directed line segment $\overrightarrow{p_0c_j}$ has one of the following properties for all $\overrightarrow{p_0c_l}$, $l \neq j$: *either* $\overrightarrow{p_0c_j}$ is in clockwise direction compared to $\overrightarrow{p_0c_l}$, *or* p_0, c_j, and c_l are collinear and $\overline{p_0c_l} \subset \overline{p_0c_j}$.

Moreover, we need an other vertex of K (denoted later by e_u) as a result of a similar process but considering p_{k+1} and counter clockwise orientation. To solve these subproblems, we invoke the basic element of a general method for generating convex hull sets (applied of course with interval arithmetic): consider e.g. $\overrightarrow{p_0c_j}$ and $\overrightarrow{p_0c_l}$ as two vectors in the *3-dimensional space* fitting to the plane $z = 0$. Evaluate the interval inclusion of the *third component of the vector product* $(c_j - p_0) \times (c_l - p_0)$ with point interval arguments and arithmetic operations. Let $P(\overrightarrow{p_0c_j}, \overrightarrow{p_0c_l}) \in \mathbb{I}$ denote this inclusion. Now,

(i) if $\overline{P}(\overrightarrow{p_0c_j}, \overrightarrow{p_0c_l}) < 0$, then it is ensured that $\overrightarrow{p_0c_j}$ is located in counter clockwise direction compared to $\overrightarrow{p_0c_l}$;
(ii) if $\underline{P}(\overrightarrow{p_0c_j}, \overrightarrow{p_0c_l}) > 0$, then it is ensured that $\overrightarrow{p_0c_j}$ is located in clockwise direction compared to $\overrightarrow{p_0c_l}$;

(iii) if $P(\overrightarrow{p_0c_j}, \overrightarrow{p_0c_l}) = [0, 0]$, then it is ensured that p_0, c_j and c_l are collinear. If this is the case, but $\overline{p_0c_l} \subset \overline{p_0c_j}$ cannot be proved, then we define the convex hull to be undetermined;
(iv) in all the other cases we define the convex hull to be undetermined.

If the condition in Step 2 of Algorithm 4 holds, then the line segment $\overline{p_0p_{k+1}}$ is defined. If the condition of Step 3 is false or K' is undetermined by the above, then we try to produce a solution polygon in Steps 8 to 10 using p_2 and p_{k-1} instead of P_1 and P_k.

If $p_0 = p_{k+1}$, then L is undefined. In this case (Steps 11 to 13) the set $\{d_2, \ldots, d_{s-k+1}\}$ should necessarily be empty (due to the invariance property of R_i). In Step 12 we test a property similar to the function $Separate()$: if p_0 is guaranteed to be outside of K, then try to determine the convex hull of p_0 and K by a similar process as it was introduced for Step 4.

Finally, if the creation of R_i' was not possible, in Step 14 we return the original R_i active region.

Implementation of Step 10 of Algorithm 2: Notice, that this step can be realized as a special case of Algorithm 4: we have to determine the convex hull of a point p_2 and the inclusion rectangle P_1. Obviously, Steps 11 to 14 of Algorithm 4 implements this correctly with $K := P_1$.

3 The correctness of Algorithms 1 and 2

Theorem 2 *The interval implementation of Algorithm 2 eliminates only those points from R_i, which are guaranteed to be at a distance less than $\tilde{f}_0$ from all points of R_j.*

Proof. At first, notice that the result polygons satisfy the required invariance property: the initial R_i (Algorithm 1, Step 1) is either a point or a line segment or a rectangle. Assume that the input polygon R_i of Algorithm 2 satisfies the invariance property. If R_i is a point then the output of Algorithm 2 is either a point or an empty polygon (Algorithm 2, Steps 4 and 5). When R_i has more than one node, one can easily see that the separation tests and the properties of obtaining convex hulls described in Algorithm 4 guarantee that the edges of the output polygon can only touch in the required way. (Recall, that we allow concave shapes as result polygons.)

Let $R_i = R_i(b_1, \ldots, b_s)$ be the input polygon of Algorithm 2 (satisfying the invariance criterion) and let R_i' be the output polygon produced by the interval version of Algorithm 2. Consider the following cases:

(1) R_i' is empty. This can be resulted in only by Step 4 of Algorithm 2, i.e. when all nodes of R_i obtain the flag '$-$'. Due to the reliable marking of the vertices, this statement holds only if the exact computations also provide the '$-$' flag for all vertices, thus, when the whole polygon can be deleted by Theorem 1.

(2) $R_i' = R_i$. The interval algorithm variant returns this result either if all vertices of R_i are labeled with '+' (Algorithm 2, Step 5), or if a 'safety

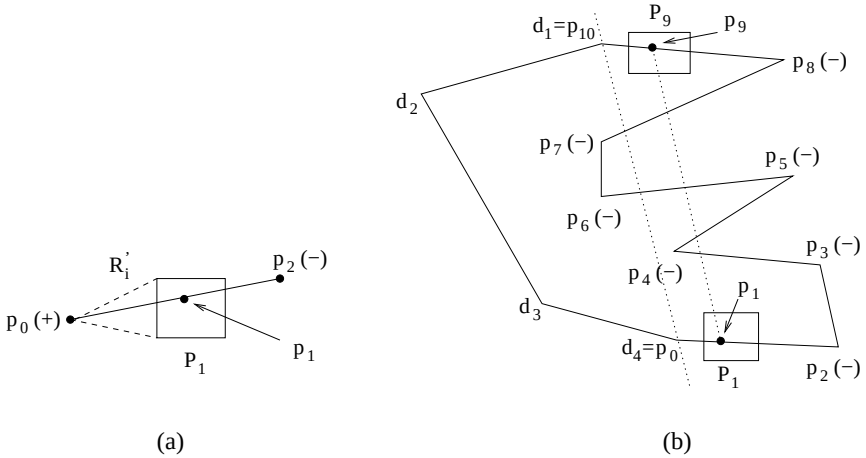


Fig. 5 Proof of correctness of Theorem 2. Case (3b) for $s = 2$ (left) and case (3c) for $s \geq 3$ with $s = 11,\ k = 9$ (right).

solution' is produced due to the overestimation or due to some technical difficulties (see Algorithms 4 and 5). This time we do not erase any possible inactive points from R_i.

(3) $R_i' \neq R_i$ and R_i' is not empty. We investigate this case for different values of s:

(3a) The case $s = 1$ cannot occur here since it is covered by the parts (1) and (2) of the proof.

(3b) If $s = 2$, we perform Steps 11 to 14 of Algorithm 4 during Step 10 of Algorithm 2, as it was discussed above. In these steps we find that $p_0 \notin P_1$ and then successfully determine R_i' as the convex hull of a point p_0 and a rectangle P_1 (see Figure 5/a). Here P_1 is either a rectangle (with $p_0, p_2 \notin P_1$) or $P_1 \equiv p_2$ (a 'safety solution') produced by Algorithm 3, and it is ensured, that P_1 contains a point p_1 which is at a distance less than $\tilde{f}_0$ from R_j. Thus, only the line segment $\overline{p_0 p_1}$ must belong to the remaining region. Clearly, this holds by the definition of the convex hull.

(3c) If $s \geq 3$, we perform Algorithm 4 as Step 17 of Algorithm 2. Consider the input polygon R_i with a nonempty set of consecutive vertices $p_2, \ldots, p_{k-1}$ labeled by '$-$'. The remaining nonempty set of nodes are denoted by $d_1 = p_{k+1}, d_2, \ldots, d_{s-k+2} = p_0$. Here $p_0 = p_{k+1}$ is also possible (see Steps 11 to 14 of Algorithm 4).

By our assumption, P_1 and P_k are successfully generated by Algorithm 3. Consequently, P_1 is either a rectangle (with $p_2 \notin P_1$) or $P_1 \equiv p_2$, and it is guaranteed, that P_1 contains a point p_1 which is at a distance less than $\tilde{f}_0$ from R_j. Obviously, similar statements hold for P_k (with p_{k-1}).

Moreover, the required separation properties tested in Algorithm 4 are satisfied (implying $p_0 \notin P_1$ and $p_{k+1} \notin P_k$, respectively) and a reliable convex hull of the set $\{p_0, p_{k+1}, P_1, P_k\}$ is determined. Figure 5/b shows an example demonstrating the case $s \geq 3$.

Denote P_+ the polygon determined by its consecutive vertices $p_1, p_k, d_1, \ldots, d_{s-k+2}$. Similarly, denote P_- the (general) polygon determined by the consecutive vertices $p_1, p_2, \ldots, p_k$. At first, P_+ satisfies the invariance property (since p_1 and p_k are separated from the half plane containing all the d_j points) and by the construction of Algorithm 4 $P_+ \subseteq R_i'$ holds. Secondly, consider a point $p \in R_i,\ p \notin P_+$. Notice, that P_- can be a self-intersecting polygon, however, in P_- only $\overline{p_1 p_k}$ can cross some other edges (by the invariance property of R_i). Thus, p must be located in one of the pieces determined by the possible intersecting points and the vertices of P_-. Nevertheless, $p \in Conv(P_-)$ must hold where $Conv(P_-)$ denotes the convex hull of P_-. We obtained

$$R_i \subseteq P_+ \cup Conv(P_-) \subseteq R_i' \cup Conv(P_-). \tag{4}$$

Assume now that a point $p \in R_i$ is eliminated by the interval implementation of Algorithm 2, i.e., $p \notin R_i'$. Then $p \in Conv(P_-)$ by (4). Recall, that $p_2, \ldots, p_{k-1}$ have the flag '$-$', and that p_1 and p_k can also receive '$-$' by definition. By Theorem 1 this means that $Conv(P_-)$ can fully be eliminated (assuming exact computations). In other words, p is eliminated by the interval method correctly. □

Corollary 1 *Algorithm 1 deletes only those* $(x, y) \in \mathbb{R}^{2n}$ *feasible points for which* $f_n(x, y) < \tilde{f}$ *holds.*

Proof. Consider the remaining regions $R_k,\ k = 1, \ldots, n$ at any time while executing Algorithm 1, and assume that $(x_i', y_i') \in R_i$ is deleted by Algorithm 2, i.e. (x_i', y_i') is at a distance less than f_0 from an $R_j, j \neq i$ region. This means that we delete all the feasible solutions $(x, y) \in [0, 1]^{2n}$, for which $(x_i, y_i) = (x_i', y_i')$ and $(x_k, y_k) \in R_k,\ \forall k = 1, \ldots, n,\ k \neq i$ holds. By Theorem 2, the distance between (x_i, y_i) and (x_j, y_j) is less than $\tilde{f}_0$, thus, from (3), the squared distance between them is guaranteed to be less than $\tilde{f}$. Consequently, $f_n(x, y) \leq (x_i - x_j)^2 + (y_i - y_j)^2 < \tilde{f}$, which completes the proof. □

4 Summary

We have introduced an area elimination method designed for the problem of finding the densest packings of equal circles in a square. Our algorithm is fully based on reliable, interval arithmetic computations. The procedure was applied as an accelerating device in our recent interval branch–and–bound global optimization algorithm [8], and it had a fundamental role in solving the earlier open problem instances of packing 28, 29, and 30 circles in the unit square.

5 Acknowledgments

This work was supported by the Grants OTKA T 032118, T 034350, T 048377, and T 046822, Grants OMFB D-30/2000, and by OMFB E-24/2001.

References

1. C. de Groot, M. Monagan, R. Peikert, and D. Würtz, *Packing circles in a square: review and new results*, in P. Kall (ed.): System Modeling and Optimization (Proc. 15th IFIP Conf. Zürich, 1991), Lecture Notes in Control and Information Services **180** (1992), pp. 45–54.
2. R. Hammer, M. Hocks, U. Kulisch, and D. Ratz, *Numerical Toolbox for Verified Computing I.*, Springer-Verlag, Berlin, 1993.
3. E. Hansen, *Global Optimization Using Interval Analysis*, Marcel Dekker, New York, 1992.
4. O. Knüppel, *PROFIL – Programmer's Runtime Optimized Fast Interval Library*, Bericht 93.4., Technische Universität Hamburg-Harburg, 1993.
5. M. Locatelli and U. Raber, *Packing Equal Circles in a Square: a Deterministic Global Optimization Approach*, Discrete Applied Mathematics 122 (2002), pp. 139–166.
6. M.Cs. Markót, *An Interval Method to Validate Optimal Solutions of the "Packing Circles in a Unit Square" Problems*, Central European Journal of Operations Research **8** (2000), pp. 63–78.
7. M.Cs. Markót, *Optimal Packing of 28 Equal Circles in a Unit Square — the First Reliable Solution*, Numerical Algorithms **37** (2004), pp. 253–261.
8. M.Cs. Markót and T. Csendes, *A New Verified Optimization Technique for the "Packing Circles in a Unit Square" Problems*, SIAM J. Optimization **16** (2005), pp. 193–219.
9. M.Cs. Markót, T. Csendes, and A.E. Csallner, *Multisection in Interval Methods for Global Optimization II. Numerical Tests*, Journal of Global Optimization **16** (1999), pp. 219–228.
10. R.E. Moore, *Interval Analysis*, Prentice–Hall, Englewood Cliffs, 1966.
11. K.J. Nurmela and P.R.J. Östergård, *Optimal packings of equal circles in a square*, in Y. Alavi, D.R. Lick, and A. Schwenk (eds.): Combinatorics, Graph Theory, and Algorithms (Proc. 8th Quadrennial International Conference on Graph Theory, Combinatorics, Algorithms, and Applications, 1999), pp. 671–680.
12. K.J. Nurmela and P.R.J. Östergård, *More Optimal Packings of Equal Circles in a Square*, Discrete and Computational Geometry **22** (1999), pp. 439–457.
13. Ratschek H. and Rokne J., *New Computer Methods for Global Optimization*, Ellis Horwood, Chichester, 1988.
14. P.G. Szabó, *Some New Structures for the "Equal Circles Packing in a Square" Problem*, Central European Journal of Operations Research **8** (2000), pp. 79–91.