A NEW VERIFIED OPTIMIZATION TECHNIQUE FOR THE "PACKING CIRCLES IN A UNIT SQUARE" PROBLEMS

MIHÁLY CSABA MARKÓT* AND TIBOR CSENDES†

Abstract. The paper presents a new verified optimization method for the problem of finding the densest packings of non-overlapping equal circles in a square. In order to provide reliable numerical results, the developed algorithm is based on interval analysis. As one of the most efficient parts of the algorithm, an interval-based version of a previous elimination procedure is introduced. This method represents the remaining areas still of interest as polygons fully calculated in a reliable way. Currently the most promising strategy of finding optimal circle packing configurations is to partition the original problem into subproblems. Still as a result of the highly increasing number of subproblems, earlier computer-aided methods were not able to solve problem instances where the number of circles was greater than 27. The present paper provides a carefully developed technique resolving this difficulty by eliminating large groups of subproblems together. As a demonstration of the capabilities of the new algorithm the problems of packing 28, 29, and 30 circles were solved within very tight tolerance values. Our verified procedure decreased the uncertainty in the location of the optimal packings by more than 700 orders of magnitude in all cases.

Key words. interval arithmetic, branch–and–bound method, circle packing, optimality proof

AMS subject classifications. 52C15, 52C26, 65G30, 65K05, 90C30

1. Introduction.

The so-called 'optimal packing of equal circles into a square' problem class is an interesting part of geometrical optimization. Circle packing has several real-life applications, e.g. cutting out a given number of identical circle shaped objects from some kind of material with a minimal amount of waste. In addition, proving the optimality of a packing configuration is a serious theoretical challenge from both the mathematical and the computational points of view.

The paper is organized as follows: in §2 we discuss the general definition of the problem and give a brief historical overview focusing mostly on those of the computer-aided methods which form the basis of the present algorithm. Section 3 contains the basic definitions and properties of interval analysis. In §4 a general interval branch–and–bound frame algorithm is introduced. After that, crucial parts of the B&B algorithm are discussed in detail, as those designed specifically for circle packing problems. As a specification step, §5 introduces a new elimination procedure based on a prior non-interval one known from the literature. In §6 the questions of finding global solutions are discussed and new methods for eliminating tile combinations are investigated. In §7 we propose an efficient way of handling occurrences of free (not fixed) circles in optimal packings. Finally, in §8 the results of the previous sections are applied by solving the packing problems of 28, 29, and 30 circles. The presented numerical results demonstrate how the algorithm works in the successive elimination steps.

2. Problem definition and history.

The basic problem we consider is the following: *place a given number n of equal circles without overlapping into a unit square maximizing the diameter of the circles as the objective function.* With a slight modification, the problem can be formulated in the following way: *place a given number n of points into the unit square maximizing the minimal distance between*

*University of Szeged, Institute of Informatics, H-6701 Szeged, P.O. Box 652, Hungary, and European Space Agency, ESTEC, Keplerlaan 1, 2201 AZ Noordwijk, The Netherlands.
†University of Szeged, Institute of Informatics, H-6701 Szeged, P.O. Box 652, Hungary.

the pairs of points. It can be shown that for a fixed $n \geq 2$ the above two problems are equivalent in the sense that there is a bijective mapping between the feasible solutions of the circle and point packing problems, and moreover, there exists a strictly monotonically increasing function transforming the objective function of the circle packing problem into the respective objective function of the point packing problem. The proof of the above statements is based on simple geometric transformations (for details see e.g. [23]). This means that an optimal solution of the circle packing problem is determined by an optimal solution of the point packing problem, and vice versa. Thus, we can consider the more simple point packing problem:

$$\text{maximize} \min_{1 \leq i \neq j \leq n} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \tag{2.1}$$

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

where the unit square is $[0,1]^2$, and the ith point is located at (x_i, y_i). The integer $n \geq 2$ is a parameter of the problem class, thus, one can refer to a particular point packing problem instance by specifying n. (Note, that we often call the problem still as circle packing.)

Due to the monotonicity of the square root function, in practice we solve the problem of maximizing

$$f_n : \mathbb{R}^{2n} \to \mathbb{R}, \quad f_n(x, y) = \min_{1 \leq i \neq j \leq n} (x_i - x_j)^2 + (y_i - y_j)^2, \tag{2.2}$$

with $0 \leq x_i, y_i \leq 1$, $i = 1, 2, \ldots, n$, saving the evaluation of the interval square root operations. In this way we compute distance values only if they are explicitly required.

Until now, the optimal packings of $2, \ldots, 9, 14, 16, 25$ and 36 circles are proved in a theoretical way. On the other hand, computer-assisted optimality proofs exist for $n \leq 20$ [5, 6, 20] and for $21 \leq n \leq 27$ [19]. Recently, [11] reported that the objective function value of the currently known best packings are correct within the tolerance value of 10^{-5} for $n = 10, \ldots, 35, 37, 38$ (without determining the location of all the optimizers). These computer methods use floating point arithmetic and bound rounding errors only during the geometric steps of the algorithms. In contrast, the present paper introduces a fully interval arithmetic based procedure providing the enclosures of both the possible optimizers and the optimum values with high accuracy. The source code and the control scripts of the algorithm are available at `http://www.inf.u-szeged.hu/~markot/packcirc.htm`.

3. Interval analysis. Our algorithm uses interval computations to produce reliable numerical solutions. This requires a brief survey on the basic interval definitions and properties (for more details see [8, 16, 21]):

The set of compact *intervals* is denoted by $\mathbb{I}$, where for all $A \in \mathbb{I}$ intervals $A = [\underline{A}, \overline{A}] = \{a \in \mathbb{R} \mid \underline{A} \leq a \leq \overline{A}\}$. Here $\underline{A}, \overline{A} \in \mathbb{R}$ mean the *lower* and *upper bounds* of A, respectively. In case of $\underline{A} = \overline{A}$ we call A a *point interval.* For a given set of reals $D \subseteq \mathbb{R}$, $\mathbb{I}(D)$ denotes the set of all intervals in D. The *width* of an interval is defined by $w(A) := \overline{A} - \underline{A}$.

The real *arithmetic operations* can be extended for intervals by applying the general definition $A \circ B := \{a \circ b \mid a \in A,\ b \in B\}$.

Let $\varphi : D \subseteq \mathbb{R} \to \mathbb{R}$ be a real *elementary function* which is continuous on all $A \in \mathbb{I}(D)$. The interval extension of φ is defined by $\Phi : \mathbb{I}(D) \to \mathbb{I}$, $\Phi(A) := \{\varphi(a) \mid a \in$

$A\}$. The interval extension of a given elementary function can be determined e.g. by invoking monotonicity properties.

A vector of n intervals is called an *n-dimensional interval* (or shortly, a *box*): $X = (X_1, X_2, \ldots, X_n)$, $X \in \mathbb{I}^n$, and $X_i \in \mathbb{I}$ for $i = 1, 2, \ldots, n$. Moreover, for a given set of n-dimensional vectors $D \subseteq \mathbb{R}^n$, let $\mathbb{I}(D)$ denote the set of n-dimensional boxes in D. The extension of operations and functions for multidimensional intervals is defined componentwise, similarly as for real vectors.

In order to define interval extensions for compound real functions, we define the *interval inclusion functions*. We call $F : \mathbb{I}(D) \to \mathbb{I}$ an *inclusion function* of $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$, if $f(X) = \{f(x) \mid x \in X\} \subseteq F(X)$ holds for all $X \in \mathbb{I}(D)$. In the previous definition $f(X)$ denotes the range of f over X. One of the possible ways of constructing such interval functions is the so-called *natural interval extension*: in the real-type function expression the variables are replaced by intervals, and moreover, the operators and elementary functions are replaced by the corresponding interval ones.

Beyond the theoretical reliability of the interval computations, the inclusion properties should be provided even in cases when a finite precision floating-point computer arithmetic is used. Namely, non-representable intervals should be handled to control rounding errors. This task is usually done by the computational environment using exactly representable floating-point numbers (also called machine numbers) and applying directed outward rounding procedures.

4. An interval branch–and–bound algorithm. In this section an interval branch–and–bound method is introduced for computing all the global maximizers and the f^* maximum value of the global optimization problem

$$\max_{z \in Z_0} f(z), \tag{4.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a continuous objective function and $Z_0 \in \mathbb{I}^n$ is the search interval. In the algorithm the interval inclusion function $F(Z)$ of $f(z)$ is used. At each iteration cycle (between Step 2 and Step 11 of Algorithm 1), choose a box Z from the boxes stored in the 'WorkTree', and split it into two parts U^1 and U^2 (Step 5). Then for both U^i try to delete some parts of U^i which cannot contain a global optimizer point (Steps 7 to 9). If the remaining part of U^i (denoted also by U^i in the algorithm) fulfills the termination criterion, put it into the ResultList (Step 10). Otherwise store the remaining part of U^i for further splittings (Step 11).

In order to avoid repeated interval function evaluations, each box Z is stored together with $\overline{F}(Z)$. The ResultList is implemented as a single-linked list and is organized according to the FIFO insertion strategy. The boxes still to be processed are placed in a balanced binary search tree, namely in an AVL-tree called WorkTree. The tree elements are sorted in decreasing order of their $\overline{F}(Z)$ values. Moreover, elements having the same $\overline{F}(Z)$ values are stored at the same node of the tree in a single-linked list. The latter list type has the same properties as the ResultList. The search tree representation offers *Insert()* for insertion, *Delete()* for deletion and *Head()* for returning the first element stored in the tree. That is, *Head()* implements an *interval selection method* choosing a box with maximal $\overline{F}(Z)$ for subdivision: this method is also known as the Moore–Skelboe selection rule.

In the following, we specify some further details of the algorithm such as the evaluation of $F(Z)$, the bisection strategy (Step 5), and the accelerating devices (Step 8).

Algorithm 1 *Global_Optimization*

Inputs:
- f: the objective function,
- Z_0: the search box,
- ε: tolerance value for the stopping criterion.

Outputs:
- $Maximum$: enclosure of the global maximum value,
- $ResultList$: set of candidates for a global maximizer.

1: $Z := Z_0$; $WorkTree := \{(Z, \overline{F}(Z))\}$; Set an initial $\tilde{f}$ cutoff value.
2: **while** ($WorkTree$ is not empty) **do**
3: $(Z, \overline{F}(Z)) := Head(WorkTree)$;
4: $Delete(Head(WorkTree))$;
5: $Bisection(Z, U^1, U^2)$;
6: **for** $i := 1$ **to** 2 **do**
7: Try to improve $\tilde{f}$;
8: Apply accelerating devices for U^i;
9: **if** (U^i can be deleted as a whole) **then continue** with the next i;
10: **if** $(w(F(U^i)) < \varepsilon)$ **then** $Insert(ResultList, (U^i, \overline{F}(U^i)))$;
11: **else** $Insert(WorkTree, (U^i, \overline{F}(U^i)))$;
12: $Maximum := (\tilde{f}, \max\{\overline{F}(Z) \mid (Z, \overline{F}(Z)) \in ResultList\})$;
13: **Return** $Maximum$, $ResultList$;

An interval inclusion function for the packing circles problems. In [12] an inclusion function was already given:

THEOREM 4.1. [12]: *Let $(X, Y) \subseteq [0, 1]^{2n}$, and let $D_{ij} = (X_i - X_j)^2 + (Y_i - Y_j)^2$ for all $i, j \in \{1, 2, \ldots, n\}$, $i \neq j$. Then an inclusion function of $f_n(x, y)$ over the $2n$-dimensional box (X, Y) is given by*

$$F_n(X, Y) := [\min_{1 \leq i \neq j \leq n} \underline{D}_{ij}, \min_{1 \leq i \neq j \leq n} \overline{D}_{ij}].$$

Bisection step. In Step 5 we split the leading box Z perpendicular to its widest component. If two or more components have the same width, we choose the one having the smallest index. This is the classical subdivision method. The investigation of some more sophisticated rules (see e.g. [3, 15]) for circle packing problems can be a subject of a further study.

Accelerating devices. In Step 8 several tests are performed to delete some parts of U^i which cannot contain global maximizer points. In some cases the whole box can be rejected. In order to test whether the investigated regions contain a global maximizer, we assume that a guaranteed lower bound $\tilde{f}$ of the global maximum value exists. In general, e.g. $\tilde{f} := \underline{F}(Z_0)$ is computed. For practical considerations we will use the notations $\tilde{f}_0$ and $\tilde{f}$ as the lower bounds of the objective functions in (2.1) and (2.2), respectively. In the present algorithm the initial $\tilde{f}$ value was determined as follows: we performed an interval function evaluation (with natural interval extension) over a machine point representation of the currently best-known packing, and set $\tilde{f}$ as the lower bound of the result interval. The corresponding $\tilde{f}_0$ was obtained by an interval square root evaluation:

$$f_s := \sqrt{\tilde{f}} \in \mathbb{I}, \quad \tilde{f}_0 := \underline{f_s}. \tag{4.2}$$

In Algorithm 1 the following accelerating tests are applied:

(i) *Cutoff test for* U^i: After computing the inclusions $F(U^i)$ over the boxes $U^i, i = 1, 2$, U^i can be eliminated if $\overline{F}(U^i) < \tilde{f}$. Note, that if $\tilde{f}$ is improved in Step 7, then one can discard all the elements $(Z, \overline{F}(Z))$ of the WorkTree for which $\overline{F}(Z) < \tilde{f}$ holds. As the experiences show, the improvement of $\tilde{f}$ can be a very hard task for circle packing problems requiring the verified results of some sophisticated search methods (e.g. those in [1, 4, 17]). Nevertheless, in cases when the goal is to verify the optimality of a given packing, the utilization of an appropriate initial $\tilde{f}$ can be enough.

(ii) *Elimination of inactive regions within* U^i: Assume we have a validated $\tilde{f}_0$ value. Considering U^i in the form of $(X, Y) \subseteq [0,1]^{2n}$, one can consider $(X_i, Y_i) \subseteq [0,1]^2$ as a rectangle in the unit square containing the ith point to be placed. Then from each (X_i, Y_i) we can iteratively delete those points which have a distance smaller than $\tilde{f}_0$ to *all* points of another rectangle (X_j, Y_j), $i \neq j$. This procedure may result in either the shrinking or the rejection of U^i. With regard to its importance, this method will be discussed in detail in §5.

Further interval based accelerating devices designed for packing circle problems can be found in [12], but those were not included in the present algorithm.

5. Method of active areas using polygons. This accelerating test is known from the literature (e.g. [5, 18, 19]) as a part of non-interval type methods. The key idea of this method is already given in §4. At first, we outline a possible basis algorithm (Algorithm 2): 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 R_i active regions 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 2 is Step 4 in which we delete some points (forming a so-called *inactive region*) of R_i having distance smaller than $\tilde{f}$ from each point of R_j.

Algorithm 2 *Method_of_active_areas*

Inputs:	–	$\tilde{f}_0$: a validated lower bound of the global maximum of (2.1),
	–	$(X, Y) \subseteq [0,1]^{2n}$: 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** $i := 1$ to $It_{\max}$ **do**
3:    **for all** (i, j), $1 \leq i, j \leq n$, $i \neq j$ **do**
4:       $R'_i := \mathit{Diminish\_ij}((R_i, R_j), \tilde{f}_0)$; {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** $(X'_i, Y'_i) := \mathit{Rectangular\_enclosure}(R_i)$;
8: **return** (X', Y');

One crucial part of the algorithm is the representation of the intermediate active areas (i.e. the R_i regions). One can easily show that a set of points within a 2-dimensional geometric object having a distance at least $\tilde{f}_0$ from all points of another object may be nonconvex 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 [5] the initial active regions are quantized into many rectangular pieces applying splittings both in horizontal and in vertical directions and the set of eliminated and remaining pieces were representing the inactive and active point sets, respectively. In [18] 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 [19] 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 resulted in an outstanding improvement in the computational efficiency as compared to the method using rectangular approximations. Moreover, the branching step of the current B&B frame algorithm generates rectangular splittings in a moderate way, mixing the advantages of the different approaches.

The method of [19] is based on the following lemma and theorem:

LEMMA 5.1. [19]: *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 5.2. [19]: *Assume that $p_1, \dots, p_k$ are distinct points on the boundary of a polygon R_i, 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, then the points in the polygon formed by $p_1, p_2, \dots, p_k$ are at a distance less than $\tilde{f}_0$ from all points of R_j.*

We introduce a numerically reliable area reduction method based on the above assertions. Due to the length and complexity of the algorithm, we will skip the technical details, those will be presented in a forthcoming paper [14]. (Keep in mind, that we intend to present a *computer-aided proof*, that is, it is necessary to make the algorithmic details available and to perform a proof of correctness of the algorithms.)

Assuming *exact computations*, one can easily see that if the polygons $R_i, i = 1, \dots, n$ are initialized as convex sets (as it is in the current method, see Algorithm 2, Step 1), then they remain convex after each elementary reduction made by Theorem 5.2. However, 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 make some restrictions for the shape of the polygons, namely, we assume that the remaining regions satisfy the invariance criterion below (implicitly defining what we call 'polygon' in the sequel):

Invariance criterion: *during the whole running of the interval implementation of Algorithm 2, each R_i, $i = 1, \dots, n$ (called remaining region or remaining polygon) is either a point p_1, or a line segment $\overline{p_1 p_2}$, or a 'simple' polygon $(p_1, \dots, p_k)$, $k \ge 3$, i.e. a polygon with edges $\overline{p_1 p_2}, \dots, \overline{p_{k-1} p_k}, \overline{p_k p_1}$, such that each pair of edges has at most one joint point as the joint endpoint of two consecutive edges.*

Notice that the above criterion allows also nonconvex polygons to form a remaining region. Algorithm 3 shows our designed interval algorithm for implementing an elementary area reduction method. In the algorithmic description we use the notation $d(x, y)$ for the euclidean distance between two points $x, y \in \mathbb{R}^2$.

In Algorithm 3 we consider several cases depending on the number of nodes of the polygon to be reduced: $s = 1$ is handled in Steps 4 and 5, while $s = 2$ and

Algorithm 3 *Diminish_ij* – the reliable 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 global maximum of (2.1).

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 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';

$s \geq 3$ are considered in Steps 7 to 10, and Steps 11 to 17, respectively. We represent polygons commonly as a sequence of consecutive vertices, but we assume that the coordinates of the vertices are machine numbers. (We start the elimination procedure with such polygons, see Algorithm 2, Step 1.) Each execution of the interval version of Algorithm 3 results in either an empty polygon (Step 4), if we can *provide a guarantee* that each vertex of R_i is at a distance less 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.*

Note, that $p_0 = p_{k+1}$ may hold in Steps 12 and 13; in this case we construct R_i' without duplicating this point in the result polygon.

There are several crucial parts of Algorithm 3 making the procedure reliable. We give only a short overview on them, for details see [14]. The first problem to be resolved is to mark the vertices of R_i correctly, i.e. to verify the validity of the **if** condition of Step 2. The second is to compute inclusion rectangles for some appropriate p_1 and p_k points (Steps 9, 14, 15) we would get if we used exact arithmetic. Finally, we have to build resulting polygons R_i' in Steps 10 and 17 satisfying the invariance criterion and enclosing all points we must keep.

The invariance criterion of the resulting polygon is reached basically during the latter task by testing several separation properties. Consider Figure 5.1 showing

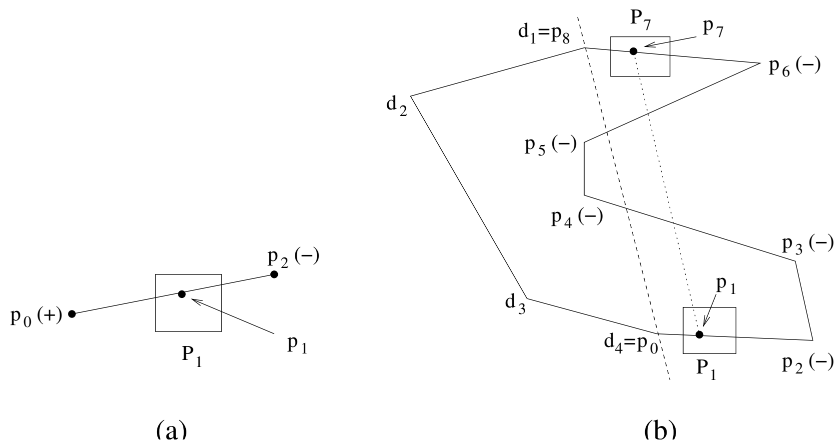

FIG. 5.1. *The polygon R_i and the enclosures P_1, P_2 in the method of active areas with interval computations. Cases $s = 2$ (left) and $s \geq 3$ with $s = 9,\ k = 7$ (right).*

an example of executing Algorithm 3 for $s = 2$ and $s \geq 3$, respectively. (Now we sketch the key idea only when $s \geq 3$, since $s = 2$ is merely a special case of this.) After creating P_1 and P_2 we can test whether the convex hull $conv(P_1, P_k)$ of them is strictly separated from the set of vertices $d_2, \ldots, d_{s-k+1}$ by the line given by p_0 and p_{k+1} (we do not allow even touching the line). If so, then the polygons formed by $d_1, \ldots, d_{s-k+2}$ and by $conv(p_0, p_{k+1}, P_1, P_k)$ both satisfy the invariance criterion. Moreover, their union (can be obtained by concatenating the corresponding vertices) also satisfies the criterion and forms a remaining polygon.

After making a more elaborative analysis on Algorithm 3, we can prove its correctness as a reliable method of eliminating inactive regions:

THEOREM 5.3. [14]: *The interval implementation of Algorithm 3 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.*

COROLLARY 5.4. [14]: *Algorithm 2 deletes only those $(x, y) \in \mathbb{R}^{2n}$ feasible points for which $f_n(x, y) < \tilde{f}$ holds.*

6. A global elimination procedure. Now we have a reliable branch–and–bound method for solving circle packing problem instances as global optimization problems. However, some difficulties would arise when one run Algorithm 1 on the whole initial search box $[0, 1]^{2n}$. The result would consist of at least $n!$ solutions showing the same packing pattern and differing from each other only in the indexing of the result components. In other words, in the optimization frame procedure the points will be permutated, but these permutations give the same solution in the geometrical sense. The other possible problem is that each solution can be transformed into symmetric, but numerically different solutions (by reflections and rotations), which also increase the number of obtained result elements unnecessarily. The predecessor [12] of the currently used interval method (which approximated the active areas by rectangular cells instead of polygons) was able to solve packing problems only up to $n = 5$ within several hours of CPU time when the permutation and symmetry problems were not resolved.

One possible solution for the above difficulties can be reached e.g. by the lexicographical ordering of the components. This was realized in an imposed form of the earlier introduced non-interval based computer-aided proofs ([5, 18, 19]). The method is called *tiling*. In the sequel we give an overview on the tiling method, and after that, we make a proposal to overcome its drawback for big number of circles and extend the optimality proof for previously unsolved problem instances.

6.1. Tiling methods. Assuming that a lower bound $\tilde{f}$ for the maximum value of the considered point packing problem instance is given, split the unit square into regions (tiles) in such a way, that the square of distance between any two points in each tile is less than $\tilde{f}$ (or the distance between any two points in each tile is less than $\tilde{f}_0$). Then for each packing configuration attaining objective function value greater than or equal to $\tilde{f}$ each tile can contain obviously at most one point of the packing. The optimality of a given packing can be proved by running the search procedure on all possible tile combinations.

REMARK 1. *For many optimization problems only an approximation or a bound of the optimum value is required by the user. We can validate such pre-given approximations when we use tiling: if all the tile combinations can fully be eliminated considering a hypothetical bound as a cutoff value, then it can be regarded as an upper bound of the global optimum.*

An interesting question arising is how to split the unit square to produce combinations which are easy to represent and handle, and in addition, to keep the number of combinations to be as small as possible. Our interval based method prefers rectangular tiles applied also by the earlier strategies. When splitting the unit square into $k \times l$ rectangles (in a regular way), the minimal number of initial tile combinations is determined by

$$\min \left\{ \binom{k \cdot l}{n} \;\middle|\; k, l \geq 1 \text{ integers}, \;\; (1/k^2 + 1/l^2)^{1/2} < \tilde{f}_0 \right\}.$$

In the previous studies the initial tile combinations were eliminated sequentially one after the other. However, the highly increasing number of those combinations made problem instances $n \geq 28$ unsolvable with this strategy in an acceptable time limit. For example, for $n = 27$ a 6×6 tiling can be applied resulting in $\binom{36}{27} \approx 9.4 \cdot 10^7$ initial time combinations. The paper [19] does not present computational details, but the optimality proof for this case was completed in about one month of CPU time (performing simple floating point computations instead of the about 4–35 times slower interval arithmetic). For $n = 28$ at least a 7×6 tiling is needed, thus, $\binom{42}{28} \approx 5.3 \cdot 10^{10}$ combinations must be checked. As in earlier studies, symmetry properties would help to reduce this number, but afterwards we would still have more than $1.3 \cdot 10^{10}$ cases.

The main idea of the proposed procedure is the following: the largest part of the combinations may be eliminated in a relatively easy way considering only local relations within the combinations. In detail, if one can discover patterns of tiles which cannot contain components of an optimal solution, then several tile combinations (i.e. higher dimensional subproblems) containing any of these patterns can be discarded quickly. Let us denote $P(n, X_1, \ldots, X_n, Y_1, \ldots, Y_n)$ a point packing problem instance where n is the number of points to be packed, $X_i, Y_i \in \mathbb{I}$, $i = 1, \ldots n$ are the components of the starting box, and the objective function is given by (2.2). The simple theorem below shows how to apply a result achieved on a $2m$-dimensional packing problem for a higher dimensional problem with $2n$ dimensions, $n \geq m \geq 2$.

THEOREM 6.1. *Let $n \geq m \geq 2$ be integers and let*

$$P_m = P(m, Z_1, \ldots, Z_m, W_1, \ldots, W_m) = P(m, (Z, W)), \text{ and}$$
$$P_n = P(n, X_1, \ldots, X_n, Y_1, \ldots, Y_n) = P(n, (X, Y))$$

be point packing problem instances $(X_i, Y_i, Z_i, W_i \in \mathbb{I}; X_i, Y_i, Z_i, W_i \subseteq [0,1])$*. Run Algorithm 1 on* P_m *using a hypothetic* $\tilde{f}$ *cutoff value in the accelerating devices and skipping Step 7, and stop after an arbitrary, preset number of iteration steps. Denote* $(Z'_1, \ldots, Z'_m, W'_1, \ldots, W'_m) := (Z', W')$ *the componentwise hull of all the elements placed on the WorkTree and on the ResultList. Assume that there exists an invertible, distance-preserving geometrical transformation* φ *with* $\varphi(Z_i) = X_i$ *and* $\varphi(W_i) = Y_i$, $\forall i = 1, \ldots, m$*. Then for each point packing* $(x, y) \in \mathbb{R}^{2n}$ *satisfying* $(x, y) \in (X, Y)$ *and* $f_n(x, y) \geq \tilde{f}$*, the statement*

$$\begin{aligned}(x, y) \in (\varphi(Z'_1), \ldots, \varphi(Z'_m), X_{m+1}, \ldots, X_n, \\ \varphi(W'_1), \ldots, \varphi(W'_m), Y_{m+1}, \ldots, Y_n) := (X', Y')\end{aligned}$$

also holds.

Proof. Perform an indirect proof and assume that a feasible solution $(x, y) \in \mathbb{R}^{2n}$ of P_n with $f_n(x, y) \geq \tilde{f}$ is discarded when modifying the starting box of P_n, i.e. $(x, y) \in (X, Y)$, but $(x, y) \notin (X', Y')$. Additionally, let $(z, w) = (\varphi^{-1}(x_1), \ldots, \varphi^{-1}(x_m), \varphi^{-1}(y_1), \ldots \varphi^{-1}(y_m)) \in \mathbb{R}^{2m}$. Notice, that $(x, y) \notin (X', Y')$ implies $(z, w) \notin (Z', W')$ by the indirect assumption, and moreover,

$$\tilde{f} \leq f_n(x, y) \leq f_m(x_1, \ldots, x_m, y_1, \ldots y_m) = f_m(z, w). \tag{6.1}$$

The second inequality of (6.1) follows from equation (2.2) and the equation in (6.1) is implied by the distance-preserving property of φ^{-1}. Consequently, Algorithm 1 deleted the feasible solution (z, w) of P_m for which $\tilde{f} \leq f_m(z, w)$ holds. But this is a contradiction, since all the accelerating devices of Algorithm 1 erase only those feasible solutions for which the objective function value is less than $\tilde{f}$. This completes the proof. □

The meaning of Theorem 6.1 is the following: assume that we are able to reduce some search regions on a tile set S'. When processing a higher dimensional subproblem on a tile set S containing the image of S', it is enough to consider *the image of those of the remaining regions of* S' as the particular components of S. Mostly we apply the theorem in the special case when φ is the identity, i.e. when the tile set of the higher dimensional problem is the superset of the tile set of the smaller problem.

As a consequence of the theorem, if it is proved that S' cannot contain point packings attaining at least $\tilde{f}$ function value, then all the higher dimensional problems with tile set S, $S' \subseteq S$ can be eliminated at once (when using the same $\tilde{f}$). The latter fact is formalized below with the notations of Theorem 6.1:

COROLLARY 6.2. *Let* φ *be the identity transformation and assume that Algorithm 1 stops with an empty WorkTree and with an empty ResultList, i.e. the whole search region* $(Z, W) = (Z_1, \ldots, Z_m, W_1, \ldots, W_m) = (X_1, \ldots, X_m, Y_1, \ldots, Y_m)$ *is eliminated by the accelerating devices using (the same)* $\tilde{f}$*. Then* (X, Y) *does not contain any* $(x, y) \in \mathbb{R}^{2n}$ *vectors for which* $f_n(x, y) \geq \tilde{f}$ *holds.*

REMARK 2. *In order to apply Theorem 6.1 correctly, we have to take care of the following important facts:*

(i) $w(Z_i) = w(X_i)$ *and* $w(W_i) = w(Y_i)$, $\forall i = 1, \ldots, m$ *should be provided. This assumption does not necessarily hold if one tries to split the unit square in a regular way using floating point numbers. The solution is the enlargement of the unit square in such a way that the resulting bounds of the tiles are exactly representable machine numbers (e.g. small integers).*

(ii) *We use shifting and rotating operators as φ, but even in these simple cases we have to apply the transformations in an interval way to obtain a guaranteed enclosure of the transformed objects.*

6.2. Basic algorithms for the optimality proofs. Our optimality proofs are based on Theorem 6.1 and have the following basic idea: first we find feasible patterns of tiles and remaining areas on some small subsets of the whole set of tiles and then we process bigger and bigger subsets while using the results of the previous steps. Thus, our method is not fully automated, it consists of several phases, and each phase depends on the result of the previous phases. (However, the whole method is easily controlled by short command scripts associated to each phase.)

Obviously, there are several ways of 'growing' the considered search regions until we reach the enclosures of the global maximizers as our final destination. We introduce two basic algorithms executed in our computer-aided proofs. We discuss them in general, i.e. for an arbitrary n and for a regular $k \times l$ splitting (with k rows and l columns) of the square. We assume that the columns are numbered by $1, \ldots, l$ from left to right. To discuss the algorithms in detail let us introduce some new notations and abbreviations:

In the beginning of each phase we determine some sets of tile combinations consisting either of the remaining areas of the previous phases (or its transformations) or of full tiles. These sets are denoted by $S^m_{s..f}$, where $s \leq f$, $s, f \in \{1, \ldots, l\}$, $m \in \{0, 1, \ldots, n\}$, symbolizing that a number of m remaining or full tile regions are considered and the regions are chosen from columns $s, s+1, \ldots, f$. (In the example of $n = 29$, we first generate $S^{15}_{1..3}$, that is, all the combinations of 15 tiles where the tiles are chosen from the leftmost 3 columns of the square.) Moreover, for a set $M = \{m_1, \ldots, m_k\} \subseteq \{0, 1, \ldots, n\}$, let $S^M_{s..f} = \cup^k_{i=1} S^{m_i}_{s..f}$. The elements of the sets $S^m_{s..f}$ are processed sequentially by the optimization algorithm using $\tilde{f}$. With the exception of the final phase, we stop the optimization algorithm after a certain number of iterations. As a result, some combinations can fully or partly be eliminated. The resulting new sets are denoted by $\bar{S}^m_{s..f}$. The sets $\bar{S}^m_{s..f}$ form the input components of the subsequent phases. We will use the notation ${}^{m_1}_{s_1..f_1}S^m_{s..f}$, $1 \leq s \leq s_1 \leq f_1 \leq f \leq l$, $0 \leq m_1 \leq m \leq n$ as that of the subset of $S^m_{s..f}$ in which each element contains exactly m_1 tile regions from columns $s_1, \ldots, f_1$. Note, that the elements of the sets $S^m_{s..f}$, $\bar{S}^m_{s..f}$, ${}^{m_1}_{s_1..f_1}S^m_{s..f}$, and ${}^{m_1}_{s_1..f_1}\bar{S}^m_{s..f}$ are in general only *enclosures* of the corresponding exact remaining areas.

REMARK 3. *In each phase of our optimality proof we accelerate Algorithm 1 by calling the method of active areas immediately to the initial search region. In many cases, this results in the elimination of the whole search box before doing any bisection. (Obviously, this modification has no effect on the correctness of Theorem 6.1.)*

Algorithm 4 (called *Grow*) takes the set $\bar{S}^m_{1..c}$ for a given pair (c, m), i.e. all the current remaining areas where m tiles are chosen from the first c columns and produces a new set ${}^{m}_{1..c}S^{m+i}_{1..(c+1)}$. The procedure takes all the combinations $A \in \bar{S}^m_{1..c}$ (Step 3) and adds a suitable (see Remark 4) number i of new tiles to them in all the possible ways. With the aid of the sets $\bar{S}^{n-m}_{c+1..l}$, in Step 6 we test whether pattern B can be valid in column $(c+1)$. If this is the case, each result combination A' (Step 7) can be tested further since we have additional information on the possible remaining areas in columns $2, \ldots, (c+1)$. Namely, $\bar{S}^{i+j}_{2..(c+1)}$ can be generated from the given $\bar{S}^{i+j}_{1..c}$ by applying Theorem 6.1 (performing a shifting transformation). In case there is no remaining combination with pattern P in $\bar{S}^{i+j}_{2..(c+1)}$ (tested in Step 9), we can

Algorithm 4 *Grow*: add one column

Inputs:
- n: the number of points to be packed,
- k, l: the square is divided into k rows and l columns regularly,
- c, $1 \le c \le l$: a column index,
- $\bar{S}^t_{1..c}$ for all $0 \le t \le ck$,
- $\bar{S}^t_{c+1..l}$ for all $0 \le t \le (l-c)k$, (if available),
- m, $1 \le m \le ck$: the elements of $\bar{S}^m_{1..c}$ will be extended by adding one column,
- i: the number of tiles to be added from column $c+1$.

Output:
- ${}^{m}_{1..c}S^{m+i}_{1..(c+1)}$: the output set.

```
1: Initialization: ${}^{m}_{1..c}S^{m+i}_{1..(c+1)} := \emptyset$;
2: Generate $S^i_{(c+1)..(c+1)}$ from full tiles;
3: for all ($A \in \bar{S}^m_{1..c}$) do
4:   Let $j$ be the number of regions in columns $2, \ldots, c$ of $A$;
5:   for all ($B \in S^i_{(c+1)..(c+1)}$) do
6:     if ($\bar{S}^{n-m}_{c+1..l}$ is available and there exists an element of it having the same tile
       pattern as B in column $c+1$) then
7:       Let $A'$ be the concatenation of $A$ and $B$;
8:       Let $P$ be the set of tile positions in columns $2, \ldots, (c+1)$ of $A'$;
9:       if (a $C \in \bar{S}^{i+j}_{2..(c+1)}$ exists with tile positions $P$) then
10:        Intersect $A'$ and $C$ on positions $P$ resulting $A''$;
11:        if (none of the components of $A''$ is empty) then
12:          Add $A''$ to ${}^{m}_{1..c}S^{m+i}_{1..(c+1)}$.
```

immediately ignore A' by Corollary 6.2. Otherwise we intersect A' with $C \in \bar{S}^{i+j}_{2..(c+1)}$ on the tile positions corresponding to P (Step 10), and apply Theorem 6.1 again to the result A'': we store A'' in the output set only if none of its components are empty (Steps 11 and 12).

Figure 6.1 demonstrates how the algorithm works for $k = l = 4$, $c = 2$, $m = 5$, $i = 3$ and for $j = 3$ in the main loop. In each part of the figure solid lines denote the boundaries of the considered combinations.

REMARK 4. *In practice, Algorithm 4 is performed on some relevant m values. In order to cover all the valid tile combinations on the first $c+1$ columns, we have to compute a lower m_l and an upper m_u bound on the number of tiles $m+i$ of the output, i.e. to associate a set of i values to each m. A possible choice for a lower bound is m, however, we may increase this number if we have a kind of complementary information: assume that it is guaranteed that all the point packings $x \in \mathbb{R}^{2n}$, $f_n(x) \ge \tilde{f}$ can contain at most t points in columns $(c+2), \ldots, l$. In other words, we know that $S^{t_0}_{c+2..l} = \emptyset$ (or $\bar{S}^{t_0}_{c+2..l} = \emptyset$) for each $t_0 > t$. Then m_l can be set to $\max(m, n-t)$. An upper bound on $m+i$ can be determined by $m_u := \min(m+k, n)$. Summarizing these results, Algorithm 4 must be performed to all i with $\max(0, n-t-m) \le i \le \min(k, n-m)$ for each m.*

Algorithm 5 called *Join* joins the elements of two sets of tile combinations pairwise. This algorithm also inputs a given parameter pair (c, m). It is assumed that the possible remaining regions are known when locating m points in the first c columns. Moreover, we have all the possibilities to add some points in the $(c+1)$th column

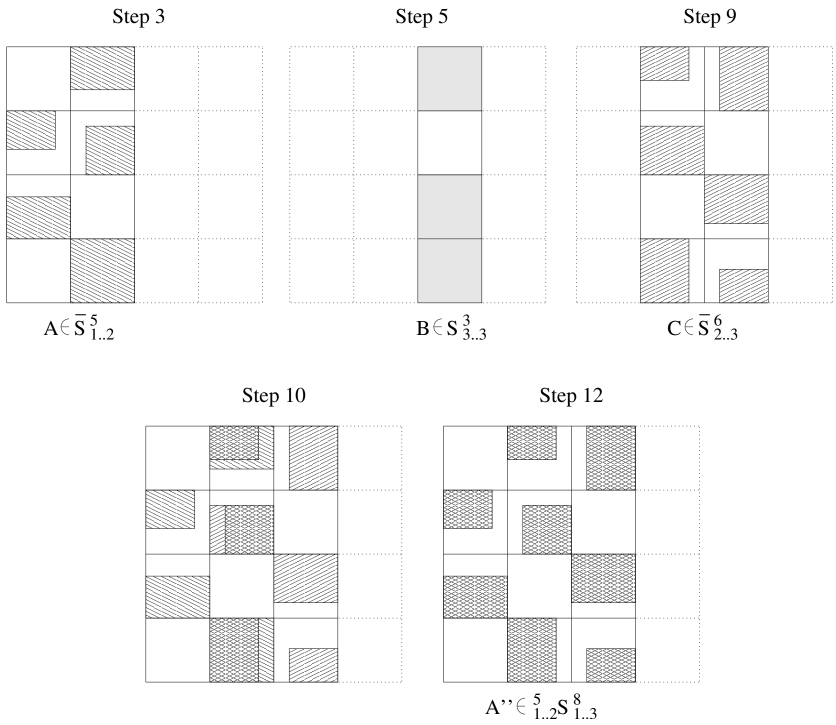

FIG. 6.1. *An example of adding a new column for $k = l = 4,\ c = 2,\ m = 5,\ i = 3,\ j = 3$.*

for all the elements of $\bar{S}^m_{1..c}$, thus, we know the sets ${}^{\;m}_{1..c}\bar{S}^{m_1}_{1..(c+1)}$, e.g. by Algorithm 4. Similarly, the sets ${}^{\;n-m}_{(c+1)..l}\bar{S}^{m_2}_{c..l}$ are also assumed to be known. (In practice, the latter sets are evaluated from ${}^{\;n-m}_{1..(l-c)}\bar{S}^{m_2}_{1..(l-c+1)}$ applying a rotation by 180 degrees around the midpoint of the search square for each element, since we 'grow' tile combinations from left to right, starting from column 1.) Consequently, the input sets can be intersected in tile positions from columns c and $c+1$.

Algorithm 5 works as follows: we take all the elements A from ${}^{\;m}_{1..c}\bar{S}^{m_1}_{1..(c+1)}$ (Step 3). Then for each A we determine *all* those combinations $C \in {}^{\;n-m}_{(c+1)..l}\bar{S}^{n-m+j}_{c..l}$ which have the same tile positions in columns c and $c+1$ as A (Steps 5 – 8). If the condition of Step 8 holds then we join A and C by Theorem 6.1 (Step 9), and finally, similarly to Algorithm 4, we store the result only if all of its components are nonempty (Steps 10, 11). The output set contains all those remaining areas of the considered packing problem for which exactly m tiles are chosen from the leftmost c columns. Figure 6.2 shows an example of executing Algorithm 5 for $n = 11,\ k = l = 4,\ c = 2,\ m = 6$ and for $m_1 = 8,\ j = 3$.

Notice that Algorithm 5 can be generalized on the base of the number i of intersecting columns. Similar representations with $i = 0, \ldots, l-1$ are possible. In the present paper only $i = 2$ is considered since it proved to be enough for the discussed problem instances. However, in general it may be worth to apply many different i values.

7. A method for handling free circles. Let us introduce the following definition:

Algorithm 5 *Join*: join remaining areas

Inputs:
- n: the number of points to be packed,
- k, l: the square is divided into k rows and l columns regularly,
- c, $1 \le c \le l-1$: a column index,
- m, $0 \le m \le ck$,
- ${}^{m}_{1..c}\bar{S}^{m_1}_{1..(c+1)}$ for all m_1, $m \le m_1 \le n$,
- ${}^{n-m}_{(c+1)..l}\bar{S}^{m_2}_{c..l}$ for all m_2, $n-m \le m_2 \le n$.

Output:
- ${}^{m}_{1..c}S^{n}_{1..l}$: the output set.

Initialization: ${}^{m}_{1..c}S^{n}_{1..l} := \emptyset$;
for $m_1 := m$ **to** n **do**
  for all $(A \in {}^{m}_{1..c}\bar{S}^{m_1}_{1..(c+1)})$ **do**
    Let j be the number of regions in column c of A;
    Let P be the set of tile positions in columns c and $(c+1)$ of A;
    for all $(C \in {}^{n-m}_{(c+1)..l}\bar{S}^{n-m+j}_{c..l})$ **do**
      Let P' be the set of tile positions in columns c and $(c+1)$ of C;
      if $(P = P')$ **then**
        Intersect A and C on positions P resulting A';
        if (none of the components of A' are empty) **then**
          Add A' to ${}^{m}_{1..c}S^{n}_{1..l}$.

DEFINITION 7.1. *Consider the point packing problem class with the distance function and objective function given by (2.2). Consider an optimal point packing, i.e. a vector* $(x,y) = (x_1,\dots,x_n,y_1,\dots,y_n) \in \mathbb{R}^{2n}$ *for which* f_n *is maximal. We call a point* $p_k = (x_k,y_k)$, $k \in \{1,\dots,n\}$ *of this optimal packing to be a* free point, *or equivalently,* a free circle centre *if there exists a half line* H *with the endpoint* p_k *and there exists a positive number* ε*, such that*

$$f_n(x,y) = f_n(x_1,\dots,x'_k,\dots,x_n,y_1,\dots,y'_k,\dots,y_n)$$

for all $(x'_k,y'_k) \in H \cap N_\varepsilon(p_k)$*, where* $N_\varepsilon(p_k)$ *denotes the* ε*-neighbourhood of* p_k*.*

Note, that by the equivalence of the point packing and circle packing problems each free point in an optimal point packing is identified by the centre of a *free circle* of the corresponding optimal packing of circles and vice versa. The meaning of the above definition is straightforward: we call a circle to be free in the optimal packing if the centre of this circle can be slightly perturbed without loosing the optimality of the packing. It is also obvious that the handling of free circles is crucial when solving circle packing problems, since they pose a continuum (and usually a positive measure) set of equivalent global optimizers.

In advanced phases of the B&B algorithm, the enclosure of such a set usually appears as a region extremely bigger than the others (and it is subdivided probably unnecessarily), but it is almost useless as a reducing region. It is apparent that one can consider only a single canonical value instead of such a point set. The following method shows how to apply a simple criterion to temporarily replace some parts of the search region with single points (without loosing any optimal solutions).

1. Let $(X_1,\dots,X_n,Y_1,\dots,Y_n) = (X,Y) \in \mathbb{I}^{2n}$ enclose all the remaining boxes (i.e. all the candidates stored either in the WorkTree or in the ResultList) after a certain number of iteration loops when executing Algorithm 1 (or its accelerated

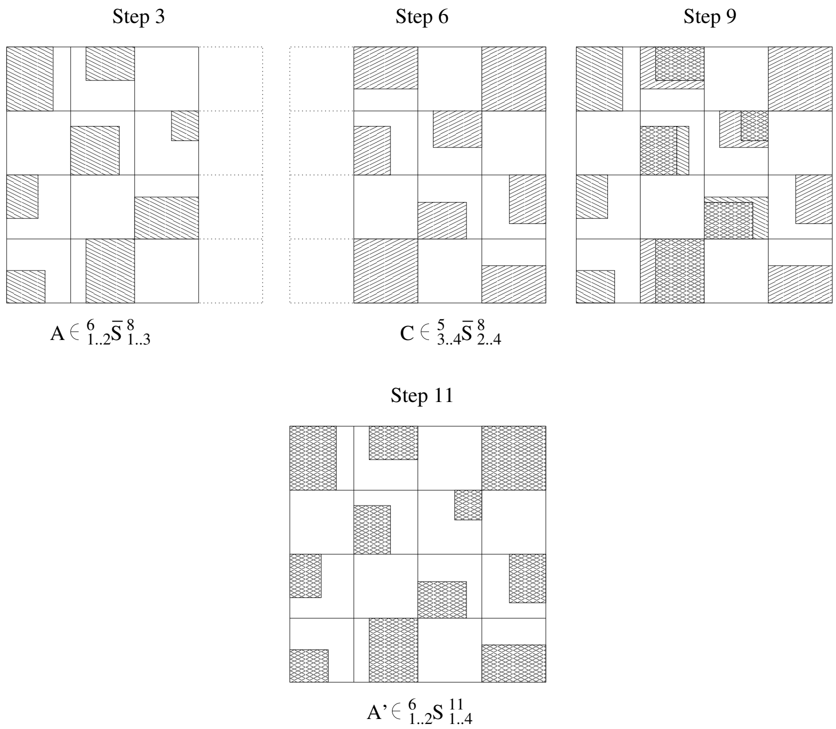

FIG. 6.2. *An example of joining the remaining areas for* $n = 11,\ k = l = 4,\ c = 2,\ m = 6$ *and for* $m_1 = 8,\ j = 3$.

version, see Remark 3). We assume that either the WorkTree or the ResultList is not empty. Moreover, let $\tilde{f}$ be the current cutoff value used in the algorithm.

2. Assume that there exist machine representable points $p_{k_1}, \ldots, p_{k_t},\ p_{k_s} \in (X_{k_s}, Y_{k_s}),\ s \in \{1, \ldots, t\}$ in t different components of (X, Y), such that for each k_s

$$\underline{D}(p_{k_s}, (X_j, Y_j)) > \overline{F}(X, Y) \geq \tilde{f} \tag{7.1}$$

holds for all $j \in \{1, \ldots, n\},\ j \neq k_s$. Let K denote the set of indices $\{k_1, \ldots k_t\}$.

3. Replace the components (X_i, Y_i) of (X, Y) with the point intervals p_i for each $i \in K$. Run Algorithm 1 on the resulting (X', Y') box ignoring the improvement step of $\tilde{f}$, i.e. using the earlier found best lower bound, and stop it after a certain number of iteration steps.

4. Let $(X'', Y'') \in \mathbb{I}^{2n}$ enclose all the remaining boxes. The components of the *output box* of the procedure are then given by (X_i, Y_i) for $i \in K$ and (X''_j, Y''_j) for each $j \notin K$, i.e. the latter components can be diminished as compared to (X_j, Y_j).

The idea behind the proposed method is the following: assume that optimal point packings exist in (X, Y). Then clearly, $\overline{F}(X, Y) \geq f^*$ holds. Consider a set of points $\{p_j \,|\, p_j \in (X_j, Y_j), j \notin K\}$ with a minimal pairwise distance f^*. Notice, that by (7.1) this set can be expanded with $\{p_i, i \in K\}$ to obtain an optimal packing. Moreover, for each $i \in K$, $d(p_i, p_j) > \overline{F}(X, Y) \geq f^*$ occurs for all $j \in \{1, \ldots, n\},\ j \neq i$, which implies the existence of an $\varepsilon > 0$ such that $d(p'_i, p_j) > f^*,\ \forall p'_i \in N_\varepsilon(p_i)$. In other words, each p_i is a free point of each such packing.

THEOREM 7.2. *The above procedure for shrinking the remaining regions is correct* *in the sense that all the optimal packings in* (X, Y) *are also present in the output box of the procedure.*

Proof. Let us use the abbreviations $Z := (X, Y)$ and $Z' := (X', Y')$. Consider a box $V' \subseteq Z'$ eliminated by Algorithm 1 using $\tilde{f}$ in Step 3 of the above method. Recall that $V_i' = p_i, \forall i \in K$. Let $V \subseteq Z$ be given by $V_j := V_j'$ for $j \notin K$ and by $V_i := Z_i$ for the other components. It is enough to show that V would also be eliminated by Algorithm 1 using $\tilde{f}$. (I.e., that V cannot contain global optimizers.)

At first, assume that V' is eliminated by the method of active areas. Notice, that from (7.1) each area reduction is made only between components $j_1, j_2 \notin K$. Clearly, if a component V_s' can fully be erased by such regions, then it could also be eliminated in V by the same test, since in V all the particular reducing regions of V' are also present.

Secondly, consider the case when V' is deleted by the cutoff test. We first prove two simple consequences of (7.1):

$$\underline{D}(p_i, Z_j) > \tilde{f} \;\Rightarrow\; \overline{D}(p_i, V_j') = \overline{D}(V_i', V_j') > \tilde{f}, \text{ and} \tag{7.2}$$

$$\underline{D}(p_i, Z_j) > \tilde{f} \;\Rightarrow\; \overline{D}(Z_i, V_j) = \overline{D}(V_i, V_j) > \tilde{f} \tag{7.3}$$

for all $i \in K,\ j \in \{1, \ldots, n\},\ i \neq j,\ V_j' \subseteq Z_j$ and $V_j \subseteq Z_j$.

Both statements can easily be proved in an indirect way applying that D is an inclusion function of the distance function d. Assuming the converse of the conclusions of (7.2) and (7.3), respectively, we could state $d(p_i, (x_j, y_j)) \leq \tilde{f}$, where (x_j, y_j) is arbitrarily chosen from V_j' for (7.2) and from V_j for (7.3). But due to $(x_j, y_j) \in Z_j$, this contradicts the premise of both (7.2) and (7.3).

Since V' is deleted by the cutoff test, we obtain

$$\tilde{f} > \overline{F}(V') = \min_{1 \leq i \neq j \leq n} \overline{D}(V_i', V_j') = \min_{\substack{1 \leq i \neq j \leq n \\ i \notin K, j \notin K}} \overline{D}(V_i', V_j') = \min_{\substack{1 \leq i \neq j \leq n \\ i \notin K, j \notin K}} \overline{D}(V_i, V_j) =$$

$$= \min_{1 \leq i \neq j \leq n} \overline{D}(V_i, V_j) = \overline{F}(V).$$

Here the conclusions of (7.2) and (7.3) are applied in the second and fourth equations, respectively, while the third equation follows from the construction of V. Thus, from the first and last statements of the chain we obtained that V can also be deleted using the same cutoff value. This completes the proof. □

According to our computational experience, the use of single values instead of regions was indispensable in achieving high precision verified solutions for the circle packing problems. The above method is performed in practice in the following way: for each component (X_i, Y_i), $i \in \{1, \ldots, n\}$ a stochastic search is applied for finding a machine representable point $p_i \in (X_i, Y_i)$ for which $g(i) = \min_{1 \leq j \leq n, j \neq i} \underline{D}(p_i, (X_j, Y_j))$ is maximal. The search procedure we used was the stochastic optimization method GLOBAL [2]. If $g(i) > \overline{F}(X, Y) \geq \tilde{f}$ holds for the best $g(i)$ value found, then it is ensured that (7.1) also holds. (Notice, that the obtained p_i is reliable since $g(i)$ is computed by interval arithmetic.)

8. Optimality for $n = 28, 29$, and 30.

8.1. Hardware and software environment. The optimization procedure was carried out on a PC having an Intel Pentium IV 1400 MHz processor and 1 Gbyte RAM, running under the Linux operating system. The global optimization frame algorithm is based on the Toolbox for C–XSC libraries [7], while the low-level interval arithmetic routines were implemented by PROFIL/BIAS [9]. Additionally, the multiple precision extension of PROFIL [10] is also built in to determine $\tilde{f}$ and to have correctly rounded decimal numbers in those I/O routines which communicate the user. (During the intermediate phases of our algorithm we used binary I/O routines.)

8.2. The global procedure. The currently known best packings for the investigated cases were published in [4] (for $n = 28$ points) and in [17] (for $n = 29$ and 30, respectively). The solution points and the corresponding function values can be found in [22], where the result components are rounded to 13 decimal digits. In order to obtain higher precision for the initial values of $\tilde{f}$, the *structure*s of the best packings were investigated. These structures describe which points are located on the sides of the square and which pairs of points have minimal distances. The structures are formally given by systems of equations. Since the exact solutions of these systems are not known for $n = 28, 29$, we solved them numerically by Maple with the precision of 20 digits. The guaranteed enclosures of the objective function values on these more precise approximate results were then determined by a simple function evaluation with multiple precision intermediate interval data.

In contrast, the coordinates of the best known packing of 30 points have the exact form of $(qd + \mathrm{par}(p+1)(1-4d), pa)$, where $d = (20-\sqrt{10})/75$, $a = 1/5$, par denotes the parity function and $p \in \{0, \dots, 5\}$, $q \in \{0, \dots, 4\}$. Moreover, the maximum value is exactly d. An enclosure of d was determined by a simple multiple precision expression evaluation.

From the above function enclosures we obtained the following $\tilde{f}_0$ lower bounds for the objective function (2.1):

$$\tilde{f}_{0,28} = 0.2305354936426673,$$
$$\tilde{f}_{0,29} = 0.2268829007442089,$$
$$\tilde{f}_{0,30} = 0.2245029645310881,$$

that allowed us to split the search space into 7×6 uniform cells as initial tiles for all the three problems.

The original point packing problem was modified in two steps: at first, we considered square of distances instead of distances between points (see §1) and then we enlarged the search space from $[0, 1] \times [0, 1]$ to $[0, 42] \times [0, 42]$ — as it was suggested by the first part of Remark 2. At the same time we evaluated the $\tilde{f}$ values

$$\tilde{f}_{28} = 93.7506267944766227,$$
$$\tilde{f}_{29} = 90.8034005467881010,$$
$$\tilde{f}_{30} = 88.9083890308478496$$

used during the whole computation.

Since the global procedure runs almost in the same way for the three problem instances, we introduce it only on the example of $n = 29$. (In [13] a similar report is given for $n = 28$.) In any case, *detailed computational results are given for all three problems* in Tables 8.1, 8.2, and 8.3, respectively. The results are grouped by the successive processing phases.

According to our previous suggestions, with the exception of the last phase we stopped the B&B algorithm after a preset number of iterations on each input combi-

TABLE 8.1

Computational details of solving the packing problem of $n = 28$ points. The two columns on the left show the name and the size of the input sets of the global optimization process: these sets are full tile combinations in Phase 1, the resulting sets of Algorithm 4 in Phase 2, the resulting sets of Algorithm 5 in Phase 3, the aggregated resulting sets of Phase 3 in Phase 4, and the only box to be refined in Phase 5, respectively. The middle group of columns contains the CPU time (CPUt, in seconds), the number of function evaluations (NFE), and the number of executions of the method of active areas (NMAA) while processing the elements of the corresponding S sets. The column set on the right shows the name and the size of the output sets of the optimization method.

S	$\lvert S\rvert$	CPUt	NFE	NMAA	$\bar{S}$	$\lvert\bar{S}\rvert$
			Phase 1			
$S^{14}_{1..3}$	116 280	958	54 846	60 249	$\bar{S}^{14}_{1..3}$	17 799
$S^{15}_{1..3}$	54 264	139	4 401	16 658	$\bar{S}^{15}_{1..3}$	1 082
$S^{16}_{1..3}$	20 349	24	27	5 253	$\bar{S}^{16}_{1..3}$	0
$S^{13}_{1..3}$	203 490	2 526	232 268	174 995	$\bar{S}^{13}_{1..3}$	77 852
			Phase 2			
${}^{13}_{1..3}S^{17}_{1..4}$	588 703	57 072	4 641 519	3 004 399	${}^{13}_{1..3}\bar{S}^{17}_{1..4}$	390 402
${}^{13}_{1..3}S^{18}_{1..4}$	632 289	47 620	2 843 863	2 214 934	${}^{13}_{1..3}\bar{S}^{18}_{1..4}$	226 564
${}^{13}_{1..3}S^{19}_{1..4}$	43 729	1 174	56 620	81 812	${}^{13}_{1..3}\bar{S}^{19}_{1..4}$	3 044
${}^{13}_{1..3}S^{20}_{1..4}$	0	-	-	-	${}^{13}_{1..3}\bar{S}^{20}_{1..4}$	0
${}^{14}_{1..3}S^{17}_{1..4}$	438 207	35 565	2 734 521	1 894 823	${}^{14}_{1..3}\bar{S}^{17}_{1..4}$	231 820
${}^{14}_{1..3}S^{18}_{1..4}$	354 167	22 466	1 312 522	1 090 923	${}^{14}_{1..3}\bar{S}^{18}_{1..4}$	103 582
${}^{14}_{1..3}S^{19}_{1..4}$	88 508	2 909	143 263	177 416	${}^{14}_{1..3}\bar{S}^{19}_{1..4}$	9 478
${}^{14}_{1..3}S^{20}_{1..4}$	2 540	13	12	2 550	${}^{14}_{1..3}\bar{S}^{20}_{1..4}$	0
${}^{14}_{1..3}S^{21}_{1..4}$	0	-	-	-	${}^{14}_{1..3}\bar{S}^{21}_{1..4}$	0
${}^{15}_{1..3}S^{17}_{1..4}$	21 844	1 276	95 333	74 306	${}^{15}_{1..3}\bar{S}^{17}_{1..4}$	7 816
${}^{15}_{1..3}S^{18}_{1..4}$	26 904	1 230	66 376	65 169	${}^{15}_{1..3}\bar{S}^{18}_{1..4}$	4 962
${}^{15}_{1..3}S^{19}_{1..4}$	14 134	312	13 370	22 543	${}^{15}_{1..3}\bar{S}^{19}_{1..4}$	853
${}^{15}_{1..3}S^{20}_{1..4}$	2 870	24	530	3 217	${}^{15}_{1..3}\bar{S}^{20}_{1..4}$	31
${}^{15}_{1..3}S^{21}_{1..4}$	0	-	-	-	${}^{15}_{1..3}\bar{S}^{21}_{1..4}$	0
${}^{15}_{1..3}S^{22}_{1..4}$	0	-	-	-	${}^{15}_{1..3}\bar{S}^{22}_{1..4}$	0
			Phase 3			
${}^{13}_{1..3}S^{28}_{1..6}$	243 762	2 147	6 910	249 347	${}^{13}_{1..3}\bar{S}^{28}_{1..6}$	56
${}^{14}_{1..3}S^{28}_{1..6}$	998 204	9 107	38 614	1 028 665	${}^{14}_{1..3}\bar{S}^{28}_{1..6}$	506
			Phase 4			
$S^{28}_{1..6}$	562	4 248	213 692	125 743	$\bar{S}^{28}_{1..6}$	6
			Phase 5			
$S^{28}_{1..6}$	1	1 854	139 334	72 982	$\bar{S}^{28}_{1..6}$	1
$\sum$	3 850 807	190 664	12 598 021	10 365 984		1 075 854

nation. This number was 3 in the first three phases and $10,000$ in the fourth (refining) phase.

Phase 1: First we have evaluated $\bar{S}^m_{1..3}$ for some m. The initial sets $S^m_{1..3}$ were built from full tile combinations, since we had no previous information about the possible configurations. Notice, that it is not worth to evaluate the above sets when m is 'small', since we may not achieve significant reductions of the active areas for such m values. On the other hand, $\bar{S}^{m_0}_{1..3} = \emptyset$ implies $\bar{S}^m_{1..3} = \emptyset$ $\forall m \geq m_0$ by Corollary 6.2. We can utilize the previous observations if we evaluate the sequence $\bar{S}^{\lceil n/2\rceil}_{1..3}, \bar{S}^{\lceil n/2\rceil+1}_{1..3}, \ldots, \bar{S}^{\lceil n/2\rceil+t}_{1..3}$ until we obtain $\bar{S}^{\lceil n/2\rceil+t}_{1..3} = \emptyset$. Then we evaluate the sets $\bar{S}^{\lfloor n/2\rfloor}_{1..3}$ (only if n is odd), $\bar{S}^{\lfloor n/2\rfloor-1}_{1..3}, \ldots, \bar{S}^{\lfloor n/2\rfloor-t+1}_{1..3}$. For $n = 29$, $\lceil n/2\rceil = 15$ and $\lfloor n/2\rfloor = 14$, thus, only $\bar{S}^{M_1}_{1..3}, M_1 = \{15, 16, 17, 14, 13\}$ was computed because $\bar{S}^{17}_{1..3}$

TABLE 8.2

Computational details of solving the packing problem of $n = 29$ points. The table is organized in the same way as Table 8.1.

S	$\|S\|$	CPUt	NFE	NMAA	$\bar{S}$	$\|\bar{S}\|$
			Phase 1			
$S_{1..3}^{15}$	54 264	305	14 928	22 758	$\bar{S}_{1..3}^{15}$	4 194
$S_{1..3}^{16}$	20 349	50	395	5 518	$\bar{S}_{1..3}^{16}$	40
$S_{1..3}^{17}$	5 985	8	0	1 574	$\bar{S}_{1..3}^{17}$	0
$S_{1..3}^{14}$	116 280	1 618	104 620	86 973	$\bar{S}_{1..3}^{14}$	33 794
$S_{1..3}^{13}$	203 490	3 454	338 771	230 121	$\bar{S}_{1..3}^{13}$	112 824
			Phase 2			
${}_{1..3}^{13}S_{1..4}^{18}$	318 042	42 945	2 660 219	1 712 120	${}_{1..3}^{13}\bar{S}_{1..4}^{18}$	221 087
${}_{1..3}^{13}S_{1..4}^{19}$	111 682	8 323	475 178	391 266	${}_{1..3}^{13}\bar{S}_{1..4}^{19}$	35 833
${}_{1..3}^{13}S_{1..4}^{20}$	0	-	-	-	${}_{1..3}^{13}\bar{S}_{1..4}^{20}$	0
${}_{1..3}^{14}S_{1..4}^{18}$	691 320	77 284	4 638 570	3 182 171	${}_{1..3}^{14}\bar{S}_{1..4}^{18}$	383 148
${}_{1..3}^{14}S_{1..4}^{19}$	248 188	18 399	1 049 402	848 878	${}_{1..3}^{14}\bar{S}_{1..4}^{19}$	80 553
${}_{1..3}^{14}S_{1..4}^{20}$	20 685	492	15 298	31 861	${}_{1..3}^{14}\bar{S}_{1..4}^{20}$	520
${}_{1..3}^{14}S_{1..4}^{21}$	0	-	-	-	${}_{1..3}^{14}\bar{S}_{1..4}^{21}$	0
${}_{1..3}^{15}S_{1..4}^{18}$	113 761	10 142	605 305	448 297	${}_{1..3}^{15}\bar{S}_{1..4}^{18}$	49 428
${}_{1..3}^{15}S_{1..4}^{19}$	73 971	4 357	241 322	216 327	${}_{1..3}^{15}\bar{S}_{1..4}^{19}$	18 064
${}_{1..3}^{15}S_{1..4}^{20}$	20 303	605	21 069	34 290	${}_{1..3}^{15}\bar{S}_{1..4}^{20}$	1 137
${}_{1..3}^{15}S_{1..4}^{21}$	652	3	2	654	${}_{1..3}^{15}\bar{S}_{1..4}^{21}$	0
${}_{1..3}^{15}S_{1..4}^{22}$	0	-	-	-	${}_{1..3}^{15}\bar{S}_{1..4}^{22}$	0
${}_{1..3}^{16}S_{1..4}^{18}$	689	33	1 839	1 742	${}_{1..3}^{16}\bar{S}_{1..4}^{18}$	130
${}_{1..3}^{16}S_{1..4}^{19}$	855	20	999	1 471	${}_{1..3}^{16}\bar{S}_{1..4}^{19}$	60
${}_{1..3}^{16}S_{1..4}^{20}$	557	4	55	601	${}_{1..3}^{16}\bar{S}_{1..4}^{20}$	0
${}_{1..3}^{16}S_{1..4}^{21}$	114	0	0	114	${}_{1..3}^{16}\bar{S}_{1..4}^{21}$	0
${}_{1..3}^{16}S_{1..4}^{22}$	0	-	-	-	${}_{1..3}^{16}\bar{S}_{1..4}^{22}$	0
${}_{1..3}^{16}S_{1..4}^{23}$	0	-	-	-	${}_{1..3}^{16}\bar{S}_{1..4}^{23}$	0
			Phase 3			
${}_{1..3}^{13}S_{1..6}^{29}$	2 349	17	67	2 400	${}_{1..3}^{13}\bar{S}_{1..6}^{29}$	1
${}_{1..3}^{14}S_{1..6}^{29}$	860 709	6 875	21 410	878 061	${}_{1..3}^{14}\bar{S}_{1..6}^{29}$	180
			Phase 4			
$S_{1..6}^{29}$	181	3 785	157 168	80 731	$\bar{S}_{1..6}^{29}$	4
			Phase 5			
$S_{1..6}^{29}$	1	3	166	107	$\bar{S}_{1..6}^{29}$	1
$\sum$	2 864 427	178 722	10 346 783	8 178 035		940 998

has proved to be empty. During the subsequent phases the sets $\bar{S}_{1..3}^{m}, m \leq 12$ were considered to be consisting of full tiles.

Acceleration of Phase 1: we reduced the necessary computational effort by filtering out symmetric cases from each set $S_{1..3}^{m}$. We applied the following transformations: reflections to the axis $x = 10.5$ and $y = 21$ and reflection to the point $(10.5, 21)$. Each full tile combination is identified by a binary string, thus, the filtering procedure can be done by simple transformations on these strings. After the elimination procedure made on $S_{1..3}^{m}$, the inverse transformations have to be applied for the remaining areas to generate the correct $\bar{S}_{1..3}^{m}$.

Phase 2: At this moment we had $\bar{S}_{1..3}^{m}$ for $0 \leq m \leq 21$. We have evaluated ${}_{1..3}^{m}\bar{S}_{1..4}^{m+i}$ for each necessary (m, i) pairs in two steps: first, we have computed ${}_{1..3}^{m}S_{1..4}^{m+i}$ by Algorithm 4 and then run the optimization algorithm on these sets to obtain ${}_{1..3}^{m}\bar{S}_{1..4}^{m+i}$.

TABLE 8.3
Computational details of solving the packing problem of $n = 30$ points. The table is organized in the same way as Table 8.1.

S	$\lvert S\rvert$	CPUt	NFE	NMAA	$\bar{S}$	$\lvert\bar{S}\rvert$
			Phase 1			
$S^{15}_{1..3}$	54 264	499	27 566	30 132	$\bar{S}^{15}_{1..3}$	7 974
$S^{16}_{1..3}$	20 349	91	1 561	6 318	$\bar{S}^{16}_{1..3}$	255
$S^{17}_{1..3}$	5 985	13	20	1 590	$\bar{S}^{17}_{1..3}$	0
$S^{14}_{1..3}$	116 280	2 248	146 498	109 752	$\bar{S}^{14}_{1..3}$	47 003
			Phase 2			
${}^{14}_{1..3}S^{18}_{1..4}$	0	-	-	-	${}^{14}_{1..3}\bar{S}^{18}_{1..4}$	0
${}^{14}_{1..3}S^{19}_{1..4}$	414 511	40 089	2 298 205	1 685 461	${}^{14}_{1..3}\bar{S}^{19}_{1..4}$	182 441
${}^{14}_{1..3}S^{20}_{1..4}$	53 774	3 110	117 194	129 588	${}^{14}_{1..3}\bar{S}^{20}_{1..4}$	7 146
${}^{14}_{1..3}S^{21}_{1..4}$	0	-	-	-	${}^{14}_{1..3}\bar{S}^{21}_{1..4}$	0
${}^{15}_{1..3}S^{18}_{1..4}$	53 490	7 289	432 161	281 500	${}^{15}_{1..3}\bar{S}^{18}_{1..4}$	37 167
${}^{15}_{1..3}S^{19}_{1..4}$	191 328	14 536	814 390	653 867	${}^{15}_{1..3}\bar{S}^{19}_{1..4}$	63 213
${}^{15}_{1..3}S^{20}_{1..4}$	58 757	3 602	133 421	141 579	${}^{15}_{1..3}\bar{S}^{20}_{1..4}$	8 812
${}^{15}_{1..3}S^{21}_{1..4}$	3 899	45	564	4 357	${}^{15}_{1..3}\bar{S}^{21}_{1..4}$	7
${}^{15}_{1..3}S^{22}_{1..4}$	0	-	-	-	${}^{15}_{1..3}\bar{S}^{22}_{1..4}$	0
${}^{16}_{1..3}S^{18}_{1..4}$	3 055	311	17 255	12 708	${}^{16}_{1..3}\bar{S}^{18}_{1..4}$	1 359
${}^{16}_{1..3}S^{19}_{1..4}$	7 170	368	19 367	18 519	${}^{16}_{1..3}\bar{S}^{19}_{1..4}$	1 359
${}^{16}_{1..3}S^{20}_{1..4}$	4 697	153	4 878	7 983	${}^{16}_{1..3}\bar{S}^{20}_{1..4}$	263
${}^{16}_{1..3}S^{21}_{1..4}$	1 186	13	206	1 342	${}^{16}_{1..3}\bar{S}^{21}_{1..4}$	7
${}^{16}_{1..3}S^{22}_{1..4}$	20	0	0	20	${}^{16}_{1..3}\bar{S}^{22}_{1..4}$	0
${}^{16}_{1..3}S^{23}_{1..4}$	0	-	-	-	${}^{16}_{1..3}\bar{S}^{23}_{1..4}$	0
			Phase 3			
${}^{14}_{1..3}S^{30}_{1..6}$	16 799	104	10	16 807	${}^{14}_{1..3}\bar{S}^{30}_{1..6}$	0
${}^{15}_{1..3}S^{30}_{1..6}$	239 430	2 068	3 197	242 002	${}^{15}_{1..3}\bar{S}^{30}_{1..6}$	36
			Phase 4			
$S^{30}_{1..6}$	36	1 391	79 916	39 935	$\bar{S}^{30}_{1..6}$	2
			Phase 5			
$S^{30}_{1..6}$	1	0	18	7	$\bar{S}^{30}_{1..6}$	1
$\sum$	1 245 031	75 930	4 096 427	3 383 467		357 045

The parameters m and i were determined in the following way (cf. Remark 4):

We had to deal only with $m \in M_2 = \{13, 14, 15, 16\}$, where M_2 was the result of Phase 1, i.e. we added one column to each element of the set $\bar{S}^{M_2}_{1..3}$. According to Remark 4, a lower bound m_l of the number of output tiles $m+i$ could be determined by a similar process as Phase 1: $\bar{S}^{12}_{5..6}$ proved to be empty (but $\bar{S}^{11}_{5..6}$ did not), thus, we had to choose at least $m_l = 18$ tiles from the first 4 columns. The upper bounds for $m+i$ were given by $m+7$ for each m.

Phase 3: It is easy to see that we could proceed with adding other columns to the result patterns of Phase 2. Instead, we have found that already at this time we had enough local information to join our remaining combinations by Algorithm 5 and to obtain a relatively small set of combinations consisting of 29 tiles.

As a result of Phase 2, we knew ${}^{i}_{1..3}\bar{S}^{m_i}_{1..4}$ for all $18 \le m_i \le 29,\ i \in M_2$. (Recall, that we did not need to consider m_i values with $i \le m_i < 18$.) Since the above sets are nonempty only when $m_{13} \in M_{13} = \{18, 19\}$, $m_{14} \in M_{14} = \{18, 19, 20\}$, $m_{15} \in M_{15} = \{18, 19, 20\}$ and $m_{16} \in M_{16} = \{18, 19\}$, respectively, it was enough to consider these m_i values when applying Algorithm 5. As it is suggested in the discussion of Algorithm 5, the other input sets ${}^{i}_{4..6}\bar{S}^{M_i}_{3..6}$, $i \in M_2$ were evaluated from ${}^{i}_{1..3}\bar{S}^{M_i}_{1..4}$ by rotating each element by 180 degrees around the midpoint $(21, 21)$ of the square.

Thus, in Phase 3 we had to join ${}^{13}_{1..3}\bar{S}^{M_{13}}_{1..4}$ with ${}^{16}_{4..6}\bar{S}^{M_{16}}_{3..6}$ (this process is called $Join(13, 16)$ in the sequel) and similarly, join ${}^{14}_{1..3}\bar{S}^{M_{14}}_{1..4}$ with ${}^{15}_{4..6}\bar{S}^{M_{15}}_{3..6}$ ($Join(14, 15)$), then join ${}^{15}_{1..3}\bar{S}^{M_{15}}_{1..4}$ with ${}^{14}_{4..6}\bar{S}^{M_{14}}_{3..6}$ ($Join(15, 14)$) and finally, join ${}^{16}_{1..3}\bar{S}^{M_{16}}_{1..4}$ with ${}^{13}_{4..6}\bar{S}^{M_{13}}_{3..6}$ ($Join(16, 13)$). The resulting sets ${}^{M_2}_{1..3}S^{29}_{1..6}$ had to be processed by the optimization algorithm sequentially. Since we stated that for a solution having objective function value greater than or equal to $\tilde{f}$ only 13, 14, 15 or 16 tiles can be chosen from each half of the search region, the union of the resulting sets

$$ {}^{13}_{1..3}\bar{S}^{29}_{1..6} \cup {}^{14}_{1..3}\bar{S}^{29}_{1..6} \cup {}^{15}_{1..3}\bar{S}^{29}_{1..6} \cup {}^{16}_{1..3}\bar{S}^{29}_{1..6} $$

can be considered as $S^{29}_{1..6}$, i.e. a set which contains all the global maximizers.

Acceleration of Phase 3: Again, symmetry properties were applied to reduce the necessary computations. Namely, without loss of generality we can assume that in each element of $S^{29}_{1..6}$ the left half of the search square contains more active regions than the right one (keeping in mind that points located on the vertical halving line segment are considered to be contained in both half squares). Consequently, we need not execute $Join(15, 14)$ and $Join(16, 13)$ when $Join(14, 15)$ and $Join(13, 16)$ are already done. Another possibility of exploiting symmetry is applied for even n numbers: for example, in $Join(14, 14)$ for $n = 28$ we assumed that column 3 does not contain less regions than column 4, i.e. $Join(14, 14)$ could be restricted to join each ${}^{14}_{1..3}\bar{S}^{i}_{1..4}$ with each ${}^{14}_{4..6}\bar{S}^{j}_{3..6}$ only when $j \geq i$. A similar acceleration was done for $n = 30$ in $Join(15, 15)$. (We presented only two very simple cases for filtering symmetric combinations, that can easily be controlled in our implementation. Obviously, one can develop more sophisticated methods resulting in larger filtering improvements.)

Phase 4: According to the previous remarks, only the two resulting sets ${}^{13}_{1..3}\bar{S}^{29}_{1..6}$ and ${}^{14}_{1..3}\bar{S}^{29}_{1..6}$ were put into a set $S^{29}_{1..6}$ to be refined. Since this set may contain a lot of equivalent solutions, we still did not use the original stopping criterion of Algorithm 1, instead, we increased the maximal number of allowed iteration steps from 3 to 10,000. As a result, we had managed to reduce the number of remaining combinations to 6 for $n = 28$, to 4 for $n = 29$ and to 2 for $n = 30$.

Phase 5: After Phase 4 it was possible to check all the remaining regions one-by-one. (However, for some n values, a much bigger number of combinations may remain, thus, we are planning to automatize this step in the future.) For the problem of packing 29 points the first two combinations were leading to symmetric packings (since the initial tile combinations of them were symmetric), and so did the other two combinations. But we found that the initial tile sets of the first group cannot be transformed to the tile sets of the second group of combinations. The reason of this is that the 7×6 splitting is not invariant to rotations by ± 90 degrees around the midpoint of the square.

However, it was easy to see that if we split the square into 6×7 regular tiles, each remaining region of one of the latter two combinations is located in different tiles of the new splitting, and the new tile set corresponding to this combination is symmetric to the patterns of the first two combinations. Consequently, it was enough to consider one of the 4 combinations in the further investigations. (One can guess that 4 combinations should exist for both groups of remaining subproblems. Indeed, it is easy to check that the additional combinations were filtered at Phase 3 by the mentioned symmetry rules.)

TABLE 8.4

The enclosure of the global maximizer point for packing of 28 points.

i	X_i	Y_i
1.	[0.0000000000000000, 0.0000000000000142]	[0.0000000000000000, 0.0000000000000182]
2.	[0.0000000000000000, 0.0000000000000723]	[0.2833357019511142, 0.2833357019512377]
3.	[0.0000000000000000, 0.0000000000000617]	[0.5138711955937813, 0.5138711955939050]
4.	[0.0000000000000000, 0.0000000000000689]	[0.7444066892364485, 0.7444066892365722]
5.	[0.0000000000000000, 0.0000000000000094]	[0.9999999999999859, 1.0000000000000000]
6.	[0.1818703764471228, 0.1818703764471709]	[0.1416678509755570, 0.1416678509756246]
7.	[0.1996495939685054, 0.1996495939687475]	[0.3986034487723758, 0.3986034487725656]
8.	[0.1996495939685047, 0.1996495939686531]	[0.6291389424150430, 0.6291389424152327]
9.	[0.1918713858351473, 0.1918713858351900]	[0.8722033446182193, 0.8722033446182866]
10.	[0.3637407528942488, 0.3637407528964056]	[0.0000000000000000, 0.0000000000015377]
11.	[0.3815199704156590, 0.3815199704157629]	[0.2569355977967746, 0.2569355977969572]
12.	[0.3992990052060252, 0.4023815131388759]	[0.5089492733445356, 0.5138712298523256]
13.	[0.3915209798036835, 0.3915209798037213]	[0.7569355977968880, 0.7569355977969508]
14.	[0.3837427716702937, 0.3837427716703746]	[0.9999999999999370, 1.0000000000000000]
15.	[0.5633903468627852, 0.5633903468641770]	[0.1152677468208971, 0.1152677468228100]
16.	[0.5839312817244451, 0.5839312817244956]	[0.3672817571470099, 0.3672817571470896]
17.	[0.5911705737722300, 0.5911705737722556]	[0.6416678509755755, 0.6416678509756131]
18.	[0.5833923656388268, 0.5833923656389046]	[0.8847322531786048, 0.8847322531787316]
19.	[0.7630399408313210, 0.7630399408318696]	[0.0000000000000000, 0.0000000000001518]
20.	[0.7694645063572478, 0.7694645063573329]	[0.2304459563257084, 0.2304459563258553]
21.	[0.7727203431310999, 0.7727203431311068]	[0.4995893688792053, 0.4995893688792224]
22.	[0.7830419596073901, 0.7830419596074357]	[0.7694645063572883, 0.7694645063573327]
23.	[0.7830419596073660, 0.7830419596074357]	[0.9999999999999551, 1.0000000000000000]
24.	[0.9935754344739882, 0.9935754344745461]	[0.0000000000000000, 0.0000000000000352]
25.	[0.9999999999999152, 1.0000000000000000]	[0.2304459563257084, 0.2304459563257429]
26.	[0.9999999999999947, 1.0000000000000000]	[0.4609814499683757, 0.4609814499684068]
27.	[0.9999999999999551, 1.0000000000000000]	[0.6915169436110431, 0.6915169436111594]
28.	[0.9999999999999316, 1.0000000000000000]	[0.9220524372537104, 0.9220524372539002]

In the cases of 28 and 30 points similar observations were made resulting in the same consequence, i.e. that one can deal only with one combination. We have obtained the first main result of our study: *Let* $n \in \{28, 29, 30\}$. *Apart from symmetric cases, one initial tile combination contains all the globally optimal solutions of the packing problem of* n *points.* We shortly improve this statement by proving that all the global solutions are located in a very small area within the particular tile combination.

As a further refinement of the only remaining box, the method for guessing free points and shrinking several components of the remaining regions was applied (see §7). We obtained that for $n = 28$ only the 12th component, for $n = 29$ only the 5th component can be replaced by a point, while for $n = 30$ no components can be replaced. (These facts are in accordance with the location of the free points of the best-known packings). The only remaining regions were then refined by Algorithm 1 using the stopping criterion parameter of $\varepsilon = 10^{-10}$. As it was also stated in [12], the cost of this local investigation was relatively small as compared to the global part of our whole procedure. Finally, after transforming the enclosures of the result boxes back to the original point packing problem, we have obtained the tightest currently known enclosures of the global maximum values, which are

TABLE 8.5
The enclosure of the global maximizer point for packing of 29 points.

i	X_i	Y_i
1.	$[\underline{0.0000000000000}000, \underline{0.0000000000000}246]$	$[\underline{0.0000000000000}000, \underline{0.0000000000000}393]$
2.	$[\underline{0.0000000000000}000, \underline{0.0000000000000}217]$	$[\underline{0.2268829007442}089, \underline{0.2268829007442}435]$
3.	$[\underline{0.0000000000000}000, \underline{0.0000000000000}199]$	$[\underline{0.4537658014884}179, \underline{0.4537658014884}444]$
4.	$[\underline{0.0000000000000}000, \underline{0.0000000000000}136]$	$[\underline{0.6806487022326}268, \underline{0.6806487022326}528]$
5.	$[\underline{0.0}000000000000000, \underline{0.0}532481705058822]$	$[\underline{0.9}073423366158249, \underline{1.0}000000000000000]$
6.	$[\underline{0.2009548756284}472, \underline{0.2009548756284}715]$	$[\underline{0.1053232576939}058, \underline{0.1053232576939}296]$
7.	$[\underline{0.2009548756284}483, \underline{0.2009548756284}689]$	$[\underline{0.3322061584381}170, \underline{0.3322061584381}365]$
8.	$[\underline{0.2009548756284}484, \underline{0.2009548756284}683]$	$[\underline{0.5590890591823}258, \underline{0.5590890591823}453]$
9.	$[\underline{0.2009548756284}486, \underline{0.2009548756284}655]$	$[\underline{0.7859719599265}345, \underline{0.7859719599265}537]$
10.	$[\underline{0.2762399917698}153, \underline{0.2762399917698}536]$	$[\underline{0.9999999999999}868, \underline{1.0000000000000}000]$
11.	$[\underline{0.4019097512568}943, \underline{0.4019097512569}189]$	$[\underline{0.0000000000000}000, \underline{0.0000000000000}215]$
12.	$[\underline{0.4019097512568}967, \underline{0.4019097512569}177]$	$[\underline{0.2268829007442}089, \underline{0.2268829007442}285]$
13.	$[\underline{0.4019097512568}968, \underline{0.4019097512569}169]$	$[\underline{0.4537658014884}179, \underline{0.4537658014884}354]$
14.	$[\underline{0.4019097512568}973, \underline{0.4019097512569}210]$	$[\underline{0.6806487022326}268, \underline{0.6806487022326}512]$
15.	$[\underline{0.5983961069856}828, \underline{0.5983961069857}054]$	$[\underline{0.1134414503720}853, \underline{0.1134414503721}164]$
16.	$[\underline{0.5983961069856}840, \underline{0.5983961069857}049]$	$[\underline{0.3403243511162}964, \underline{0.3403243511163}233]$
17.	$[\underline{0.5983961069856}844, \underline{0.5983961069857}049]$	$[\underline{0.5672072518605}076, \underline{0.5672072518605}315]$
18.	$[\underline{0.5983961069856}861, \underline{0.5983961069857}088]$	$[\underline{0.7940901526047}168, \underline{0.7940901526047}401]$
19.	$[\underline{0.5031228925140}242, \underline{0.5031228925140}623]$	$[\underline{0.9999999999999}830, \underline{1.0000000000000}000]$
20.	$[\underline{0.7948824627144}688, \underline{0.7948824627144}926]$	$[\underline{0.0000000000000}000, \underline{0.0000000000000}197]$
21.	$[\underline{0.7948824627144}706, \underline{0.7948824627144}921]$	$[\underline{0.2268829007442}089, \underline{0.2268829007442}256]$
22.	$[\underline{0.7948824627144}710, \underline{0.7948824627144}921]$	$[\underline{0.4537658014884}178, \underline{0.4537658014884}334]$
23.	$[\underline{0.7948824627144}749, \underline{0.7948824627144}919]$	$[\underline{0.6806487022326}267, \underline{0.6806487022326}419]$
24.	$[\underline{0.729082658483}5373, \underline{0.729082658483}7260]$	$[\underline{0.979554100334}8881, \underline{0.979554100335}6186]$
25.	$[\underline{0.9999999999999}812, \underline{1.0000000000000}000]$	$[\underline{0.0969672447170}968, \underline{0.0969672447171}463]$
26.	$[\underline{0.9999999999999}812, \underline{1.0000000000000}000]$	$[\underline{0.3238501454613}097, \underline{0.3238501454613}550]$
27.	$[\underline{0.9999999999999}820, \underline{1.0000000000000}000]$	$[\underline{0.5507330462055}206, \underline{0.5507330462055}635]$
28.	$[\underline{0.9999999999999}859, \underline{1.0000000000000}000]$	$[\underline{0.7776159469497}363, \underline{0.7776159469497}720]$
29.	$[\underline{0.955042424453}0482, \underline{0.955042424453}2232]$	$[\underline{0.9999999999999}683, \underline{1.0000000000000}000]$

$$
\begin{aligned}
F^*_{28} &= [\underline{0.23053549364266}73, \underline{0.23053549364267}43], \quad w(F^*_{28}) \approx 7 \cdot 10^{-15},\\
F^*_{29} &= [\underline{0.2268829007442}089, \underline{0.2268829007442}240], \quad w(F^*_{29}) \approx 2 \cdot 10^{-14},\\
F^*_{30} &= [\underline{0.22450296453108}81, \underline{0.22450296453109}03], \quad w(F^*_{30}) \approx 2 \cdot 10^{-15}.
\end{aligned}
$$

The enclosure of the particular global maximizers $(X,Y)^*_n$ are reported in Tables 8.4, 8.5 and 8.6, respectively. The precision of the components are highlighted by underscores. Our conclusions are the following: *let $n \in \{28, 29, 30\}$. Apart from symmetric cases, all the global optimizers of the packing problem of n points are located in the reported boxes having very tight components (with the exception of the components enclosing possibly free points for $n = 28, 29$). Moreover, the reported enclosures confirm that the structure of the currently known best packings are optimal within the precision of $w(F^*_n)$. In other words, all the exact global optimizers are located in $(X,Y)^*_{28}$, $(X,Y)^*_{29}$, and $(X,Y)^*_{30}$, and the exact global optima differ from the currently best known function values by at most $w(F^*_n)$.*

It is also worth mentioning that our verified procedure decreased the uncertainty

TABLE 8.6
The enclosure of the global maximizer point for packing of 30 points.

i	X_i	Y_i
1.	[0.1019881418756426, 0.1019881418756476] ,	[0.0000000000000000, 0.0000000000000026]
2.	[0.0000000000000000, 0.0000000000000026] ,	[0.1999999999999995, 0.2000000000000022]
3.	[0.1019881418756451, 0.1019881418756477] ,	[0.3999999999999989, 0.4000000000000017]
4.	[0.0000000000000000, 0.0000000000000025] ,	[0.5999999999999984, 0.6000000000000011]
5.	[0.1019881418756451, 0.1019881418756478] ,	[0.7999999999999978, 0.8000000000000006]
6.	[0.0000000000000000, 0.0000000000000045] ,	[0.9999999999999972, 1.0000000000000000]
7.	[0.3264911064067307, 0.3264911064067357] ,	[0.0000000000000000, 0.0000000000000028]
8.	[0.2245029645310881, 0.2245029645310907] ,	[0.1999999999999995, 0.2000000000000022]
9.	[0.3264911064067332, 0.3264911064067358] ,	[0.3999999999999989, 0.4000000000000017]
10.	[0.2245029645310881, 0.2245029645310905] ,	[0.5999999999999984, 0.6000000000000011]
11.	[0.3264911064067332, 0.3264911064067359] ,	[0.7999999999999978, 0.8000000000000006]
12.	[0.2245029645310881, 0.2245029645310926] ,	[0.9999999999999972, 1.0000000000000000]
13.	[0.4490059290621762, 0.4490059290621788] ,	[0.1999999999999995, 0.2000000000000021]
14.	[0.4490059290621762, 0.4490059290621786] ,	[0.5999999999999984, 0.6000000000000011]
15.	[0.4490059290621762, 0.4490059290621807] ,	[0.9999999999999972, 1.0000000000000000]
16.	[0.5509940709378189, 0.5509940709378239] ,	[0.0000000000000000, 0.0000000000000025]
17.	[0.5509940709378213, 0.5509940709378240] ,	[0.3999999999999989, 0.4000000000000016]
18.	[0.5509940709378213, 0.5509940709378240] ,	[0.7999999999999978, 0.8000000000000006]
19.	[0.7754970354689073, 0.7754970354689120] ,	[0.0000000000000000, 0.0000000000000024]
20.	[0.6735088935932643, 0.6735088935932668] ,	[0.1999999999999995, 0.2000000000000020]
21.	[0.7754970354689095, 0.7754970354689120] ,	[0.3999999999999990, 0.4000000000000015]
22.	[0.6735088935932643, 0.6735088935932666] ,	[0.5999999999999986, 0.6000000000000011]
23.	[0.7754970354689095, 0.7754970354689120] ,	[0.7999999999999981, 0.8000000000000006]
24.	[0.6735088935932643, 0.6735088935932688] ,	[0.9999999999999976, 1.0000000000000000]
25.	[0.9999999999999955, 1.0000000000000000] ,	[0.0000000000000000, 0.0000000000000023]
26.	[0.8980118581243525, 0.8980118581243549] ,	[0.1999999999999995, 0.2000000000000019]
27.	[0.9999999999999976, 1.0000000000000000] ,	[0.3999999999999991, 0.4000000000000015]
28.	[0.8980118581243525, 0.8980118581243548] ,	[0.5999999999999987, 0.6000000000000011]
29.	[0.9999999999999977, 1.0000000000000000] ,	[0.7999999999999981, 0.8000000000000006]
30.	[0.8980118581243525, 0.8980118581243570] ,	[0.9999999999999976, 1.0000000000000000]

in the location of the optimal packing by as much as

$$\log\left(w([0,1])^{2n} / \prod_{i=1}^{n} w(X_n^*) w(Y_n^*)\right),$$

i.e. more than 711, 764, and 872 orders of magnitude, respectively.

As Tables 8.1 to 8.3 show, the total time complexity of the optimality proof was approximately 53, 50, and 21 hours for the packing of 28, 29, and 30 points, respectively. Notice, that the CPU time of those parts of the method not indicated in the tables (such as the determination of m_l in Phase 2 and the execution time of Algorithms 4 and 5) took about only one additional minute for each problem instance. The total time complexities are remarkably less than the forecasted execution times (i.e. one or even more decades) of the earlier methods. The memory complexity of our procedure was small in most parts of the method, since the results of the intermediate phases can be saved to storage devices. Moreover, due to the sequential and quick processing of the tile combinations, the main optimization method required small amount of memory. In several cases, the memory requirement of Algorithms 4 and 5 was large for large set of input combinations, but however, our main memory was still enough to store all necessary combinations at one time to avoid swapping.

9. Summary and future work. We introduced a verified, interval arithmetic-based optimization method for solving circle (point) packing problems. Each part of our method is discussed through detailed algorithmic descriptions and theorems of correctness. We solved the previously unsolved problem instances of packing 28, 29, and 30 points. It was shown that the earlier found best configurations are globally optimal within very tight tolerance values. We reported the boxes enclosing the exact optimizers, and stated that the current best function value differs from the maximum by at most $7 \cdot 10^{-15}$, $2 \cdot 10^{-14}$, and $2 \cdot 10^{-15}$ for $n = 28, 29$, and 30, respectively.

Our method contained two key procedures: on the one hand, we developed a very efficient area reduction algorithm. This method was based on the interval adaptation of the earlier known polygon representation procedure. On the other hand, a very effective technique based on subsets of tile combinations was introduced for handling and eliminating groups of subproblems together. As a result, the total computational time was extremely smaller than the predicted time complexity of the earlier methods.

The basic elements of the proposed method will very probably play a role in further studies of both circle packing problems for $n \geq 31$ and the generalizations of the problem class. In the next unsolved case $n = 31$ already a 7×7 tile splitting can be applied, which will probably require an additional intermediate phase for 'growing' the dimensionality of the considered subproblems. Thus, we are planning to develop our accelerated method for eliminating tile combinations in a more general way. The first part of this work would consist of programming features, such as extending Algorithm 4 by the ability of adding new rows of tiles and modify Algorithm 5 to handle general intersecting tile positions of the combinations to be joined. Then by using this more general approach, more sophisticated multi-phase methods could be designed reducing the total cost of processing intermediate tile combinations, and opening the way for solving much harder circle packing problems and their generalizations.

REFERENCES

[1] L.G. Casado, I. García, P.G. Szabó, and T. Csendes, *Equal Circles Packing in a Square II. — New Results for up to 100 Circles Using the TAMSASS-PECS Algorithm*, in F. Giannessi, P. Pardalos, and T. Rapcsák (eds.): Optimization Theory — Recent Developments from Mátraháza, pp. 207–224, Kluwer, Dordrecht, 2001.

[2] T. Csendes, *Nonlinear parameter estimation by global optimization — efficiency and reliability*, Acta Cybernetica 8 (1988), pp. 361–370.

[3] T. Csendes and D. Ratz, *Subdivision direction selection in interval methods for global optimization*, SIAM J. Num. Anal. 34 (1997), pp. 922–938.

[4] R.L. Graham and B.D. Lubachevsky, *Repeated patterns of dense packings of equal disks in a square*, Electronic Journal of Combinatorics 3:R16 (1996).

[5] 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), pp. 45–54, Lecture Notes in Control and Information Services 180 (1992).

[6] C. de Groot, R. Peikert, and D. Würtz, *The optimal packing of ten equal circles in a square*, IPS Research Report 90-12, ETH, Zürich, 1990.

[7] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz, *Numerical Toolbox for Verified Computing I.*, Springer-Verlag, Berlin, 1993.

[8] E. Hansen, *Global Optimization Using Interval Analysis*, Marcel Dekker, New York, 1992.

[9] O. Knüppel, *PROFIL – Programmer's Runtime Optimized Fast Interval Library*, Bericht 93.4., Technische Universität Hamburg-Harburg, 1993.

[10] O. Knüppel, *A Multiple Precision Arithmetic for PROFIL*, Bericht 93.6., Technische Universität Hamburg-Harburg, 1993.

[11] M. Locatelli and U. Raber, *Packing Equal Circles in a Square: a Deterministic Global Optimization Approach*, Discrete Applied Mathematics 122 (2002), pp. 139–166.

[12] 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.

[13] M.Cs. Markót, *Optimal Packing of 28 Equal Circles in a Unit Square — the First Reliable Solution.* Accepted for publication in Numerical Algorithms. Available at `http://www.inf.u-szeged.hu/~markot/opt28.ps.gz`.

[14] M.Cs. Markót and T. Csendes, *Interval techniques for solving circle packing problems – area reduction and structural optimality.* Manuscript. Available at `http://www.inf.u-szeged.hu/~markot/intpoly.ps.gz`.

[15] 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.

[16] R.E. Moore, *Interval Analysis*, Prentice–Hall, Englewood Cliffs, 1966.

[17] K.J. Nurmela and P.R.J. Östergård, *Packing up to 50 circles in a square*, Discrete and Computational Geometry 18 (1997), pp. 111–120.

[18] 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), pp. 671–680 (1999).

[19] 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.

[20] R. Peikert, *Dichteste Packungen von gleichen Kreisen in einem Quadrat*, Elemente der Mathematik 49 (1994), pp. 17–25.

[21] Ratschek H. and Rokne J., *New Computer Methods for Global Optimization*, Ellis Horwood, Chichester, 1988.

[22] http://www.packomania.com, updated by Eckard Specht.

[23] 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.