Greedy Algorithms for Packing Unequal Circles into a Rectangular Container[†‡]

WenQi HUANG[a], Yu LI[b,*], Hakim AKEB[b], ChuMin LI[b]

[a] *College of Computer Science, HuaZhong Univ. of Science and Technology, Wuhan 430074, China,*
E-mail: wqhuang@mail.hust.edu.cn

[b] *LaRIA, Université de Picardie Jules Verne, 33 Rue Saint Leu, 80039 Amiens cedex 1, France,*
E-mail:{yu.li, hakim.akeb, chu-min.li}@u-picardie.fr

Abstract

In this paper, we study the problem of packing unequal circles into a $2D$ rectangular container. We solve this problem by proposing two greedy algorithms. The first algorithm, denoted by B1.0, selects the next circle to place according to the *maximum hole degree rule*, which is inspired from human activity in packing. The second algorithm, denoted by B1.5, improves B1.0 with a *self look-ahead search strategy*. The comparisons with the published methods on several instances taken from the literature show the good performance of our approach.

Keywords: circle packing problem; combinatorial optimization; greedy algorithm, heuristic

Introduction

The problem of packing unequal circles into a two dimensional ($2D$) rectangular container is a circle packing problem[1], encountered in some industries (textile, glass, wood, paper, etc). It consists in placing a set of unequal circles into a rectangular container without overlap. The usual objective is to maximize the material utilization and hence to minimize the wasted area. The problem is known to be NP-complete[2].

Most published research on circle packing focuses on packing equal circles into a container[3, 4, 5]. The proposed approaches were heavily influenced by the congruence of the circles.

Several authors[1, 6, 7, 8, 9] have studied the problem of packing unequal circles into a rectangular container. George et al.[1] proposed a set of heuristic rules and different combinations of these rules to pack unequal circles into a rectangle, and the best rules were a quasi-random algorithm and a genetic algorithm. Stoyan and Yaskov[8, 9] proposed a mathematical model and a solution method based on a combination of the branch-and-bound algorithm and the reduced gradient method to pack unequal circles into a strip. Hifi and M'Hallah[7] presented a Bottom-Left (BL) based genetic algorithm to solve a so-called dual of circle packing problems: cutting unequal circles on a rectangular plate.

* Corresponding author: tel: +33 3 22827872; fax: +33 3 22825412; e-mail: yu.li@u-picardie.fr

† This work is supported by "China national 913 program (G1998030600)" and "Programme de Recherches Avancées de Coopérations Franco-Chinoises (PRA SI02-04).

In this paper, we develop two greedy algorithms to pack unequal circles into a rectangular container. They are both deterministic and constructive approximate algorithms. The first greedy algorithm, denoted by B1.0, is a basic heuristic which selects the next circle to place according to the *maximum hole degree rule*, which is inspired from human activity in packing. The second greedy algorithm, denoted by B1.5, improves B1.0 with a *self look-ahead search strategy*. This work is a continuation of Huang et al.'s work for packing unequal circles into a circular container[10, 11]. We compare the performance of B1.0 and B1.5 to the published methods on several instances taken from the literature. Experimental results show the good performance of our approach.

The paper is organized as follows. In the next section, we give a formal definition of the circle packing problem. In the following two sections, we present the two new packing algorithms B1.0 and B1.5. In a later section, we analyze the complexity of B1.0 and B1.5. In the penultimate section, we present and analyze the experimental results, and compare the performance of B1.0 and B1.5 with related work in the literature. Finally we present conclusions.

Problem of packing unequal circles into a rectangular container

We consider the following circle packing optimization problem: given a set of unequal circles and a rectangular container whose width may be changed and height is fixed, find the minimal width of the container so that all the circles can be packed into the container without overlap. The associated decision problem is formally stated as follows.

Given: a rectangle of width W and height H, and a set of n circles, denoted by c_1, ..., c_n, of different radii r_1, ..., r_n. In the *2D* Euclidean space, the center of the rectangle is placed at $(0, 0)$ and its four sides, denoted by s_1, s_2, s_3 and s_4, are parallel to X and Y axes respectively, where (x_i,y_i) is the coordinate of the center of the circle c_i. The problem is to determine if there exist $2n$ real numbers $x_1, y_1, ..., x_n, y_n$ such that

$$\frac{H}{2} - y_i - r_i \geq 0, \qquad i \in \{1, ..., n\} \tag{1}$$

$$\frac{H}{2} + y_i - r_i \geq 0, \qquad i \in \{1, ..., n\} \tag{2}$$

$$\frac{W}{2} + x_i - r_i \geq 0, \qquad i \in \{1, ..., n\} \tag{3}$$

$$\frac{W}{2} - x_i - r_i \geq 0, \qquad i \in \{1, ..., n\} \tag{4}$$

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} - r_i - r_j \geq 0, \quad i \neq j \in \{1, ..., n\} \tag{5}$$

Constraints (1)-(4) state that circle c_i placed in the rectangle should not extend outside the rectangle. In other words, these constraints mean that the distances from c_i to the four sides, respectively denoted d_{i,s_1}, d_{i,s_2}, d_{i,s_3} and d_{i,s_4}, and defined as

$$d_{i,s_1} = \frac{H}{2} - y_i - r_i, \;\; d_{i,s_2} = \frac{H}{2} + y_i - r_i, \;\; d_{i,s_3} = \frac{W}{2} + x_i - r_i, \;\; d_{i,s_4} = \frac{W}{2} - x_i - r_i$$

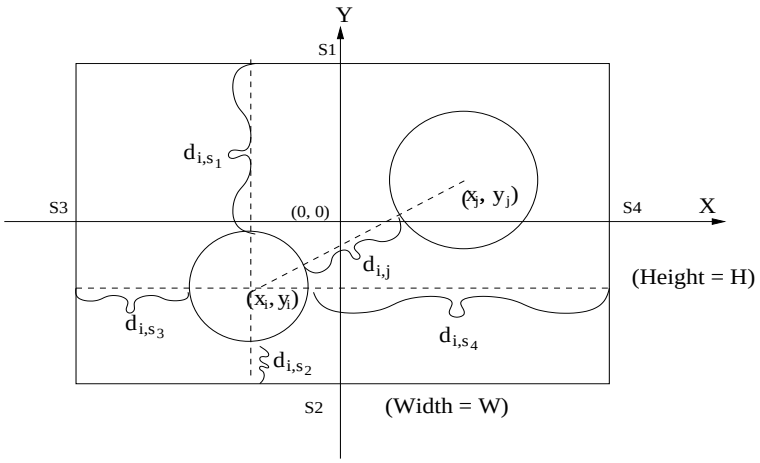

Figure 1: Problem of packing unequal circles into a rectangular container

should not be negative.

Constraint (5) requires that circles placed in the rectangle cannot overlap each other, i.e., the distance $d_{i,j}$ between two circles c_i and c_j defined as

$$d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} - r_i - r_j$$

should not be negative.

Figure 1 illustrates these constraints.

If we find an efficient algorithm to solve this decision problem, we can then solve the original optimization problem by using some search strategies. For example, we first apply dichotomous search to get rapidly a "good enough" upper bound for the width, then from this known upper bound we gradually reduce it until the algorithm no longer finds a successful configuration. The final upper bound is then taken as the minimal width of the rectangle obtained by the algorithm.

Consequently, we will concentrate our discussion on the decision problem and present two greedy algorithms to solve it in the following sections.

Maximum Hole Degree (MHD) rule and Algorithm B1.0

According to human experience in packing, the benefits of placing an object at different positions in a container are different. Particularly, this can be summarized by a Chinese proverb: "Gold corner, silver side and grass middle". It means that the corner positions possess higher value than those at the side, the side positions possess higher value than those in the middle. The notion of *corner placement* in our approach is directly inspired from these experiences and means the placement of a circle at a corner region formed by circles already placed and the container. A similar corner placement is also used in the work of Stoyan and Yaskov[8, 9], and that of Hifi and M'Hallah[7]. In addition, the well-known Bottom-Left rule, largely used to solve various packing problems, is also in conformity with

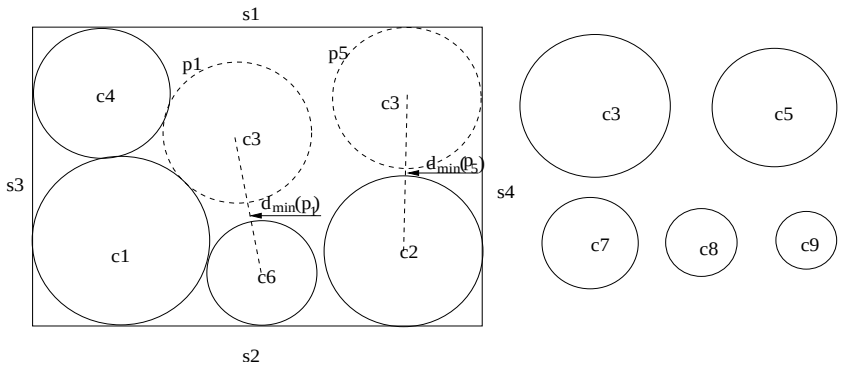

Figure 2: Two corner placements of circle c_3 for an instance of 9 circles

these experiences.

Human experience also tells us that a good placement may be such that the placed object occupies a region of a good "hole" form, i.e., the wasted area of such a region after placing the object is as small as possible. The concept of *hole degree* in our approach is a quantified measure to evaluate the benefit of a corner placement, which guides the selection of the next circle to be placed. The original idea of the *hole degree* came from the concept of *exclusive region* proposed by Huang and Sweedler[12] for solving a chip manufacture problem in VLSI. Then it was developed to pack unequal circles into a circular container by Huang et al.[10, 11]. Therefore, the above Chinese proverb might be extended to as "Diamond hole, gold corner, silver side and grass middle".

Corner placement, hole degree, and Maximum Hole Degree (MHD) rule

Definition (*Configuration*) *Let M be the set of circles already placed in the rectangle and $|M| = m$. A configuration C is a partial pattern (layout) where $m(\geq 2)$ circles have been already placed inside the rectangle without overlap, and $n - m$ circles remain to be packed into the rectangle.*

A configuration is said to be *successful* if $m = n$, i.e., all circles are placed inside the rectangle without overlap.

A configuration is said to be *fail* if $m < n$ and none of the circles outside the rectangle can be placed into the rectangle without overlapping one of the m circles already in the rectangle.

Definition (*Corner placement*) *Given a configuration C, a corner placement of a circle c_i outside the rectangle is the placement of c_i into the rectangle so that c_i touches two items without overlapping other already placed circles (an item may be a circle or one of the four sides of the rectangle). Note that the two items are not necessarily touching each other. A corner placement is represented by $p(c_i,x,y)$, meaning that c_i is to be placed at position (x, y).*

There may be several corner placements for packing a circle into the rectangle. Figure 2 illustrates an instance of 9 circles numbered in order of decreasing radius. At the configuration in Figure 2, four circles c_1, c_2, c_4 and c_6 are already placed in the rectangle. There are 6 corner placements to pack circle c_3: p_1, p_2, p_3, p_4, p_5 and p_6, touching respectively c_4 and c_1, c_1 and c_6, c_6 and c_2, c_2 and s_4, s_4 and s_1, s_1 and c_4. Figure 2 only illustrates two corner placements p_1 and p_5.

To evaluate the benefit of a corner placement, we adapt the quantified measure λ, called *legal action degree* in Huang et al.'s works to pack unequal circles into a circular container[10, 11], to define the *hole degree* notion in our approach.

Definition (*Hole degree of a corner placement*) *Let $p(c_i,x,y)$ be a corner placement, u and v be the two items (circle or side) touching circle c_i if c_i is placed at (x, y). The hole degree λ of the corner placement $p(c_i,x,y)$ is defined as*

$$\lambda = (1 - \frac{d_{min}}{r_i})$$

where r_i is the radius of c_i, and d_{min} is the minimal distance from c_i to other circles in M and sides of the rectangle (excluding u and v), i.e. $d_{min} = min_{j \in M \cup \{s_1,s_2,s_3,s_4\}, j \neq u,v}(d_{ij})$.

Figure 2 illustrates the minimal distance for both corner placements p_1 and p_5. Note that, if c_i can be packed by a corner placement into the rectangle and touches more than two items, then $d_{min} = 0$ and $\lambda = 1$; otherwise $\lambda < 1$. If λ is equal to 1, it means that the placed circle occupies a complete hole. For example, the degree λ of the placement of c_4 at the position tangent with s_1, s_3 and c_1 in Figure 2 is equal to 1, since the corresponding $d_{min} = 0$, so c_4 occupied a complete hole.

The hole degree λ of a corner placement describes how the placed circle is close to the already existing pattern. We use it as the benefit of the placement of a circle. Intuitively, since one should place a circle as close as possible to the already existing pattern, a packing procedure should do a placement with the maximum hole degree to place the next circle in the container.

Definition (*Maximum Hole Degree (MHD) rule*) *The corner placement $p(c_i,x,y)$ with the maximum degree λ is selected to place c_i at position (x, y).*

Algorithm B1.0

The MHD rule directly results in our basic greedy packing algorithm B1.0.

Given a circle packing instance denoted by I, B1.0 proceeds as follows. It starts by packing a pair of circles into the rectangle to generate an initial configuration, so that the first circle is placed at the bottom-left corner, and the second circle touches the first one and a side of the rectangle, or the second circle does not touch the first one but touches two sides of the rectangle. Under the current configuration, for each circle outside the rectangle, B1.0 generates all possible corner placements and calculates their degrees. Among these placements, B1.0 selects the placement $p(c_i, x, y)$ with the maximum degree λ, and places c_i at (x, y).

If all circles are placed in the rectangle so that constraints (1) - (5) are satisfied, B1.0 stops with success. If a failed configuration is reached, B1.0 tries the next pair of circles. If all pairs of circles have been tried but no successful configuration is found, B1.0 stops with failure.

Figure 3 illustrates a successful configuration obtained by B1.0 to pack the 9 circles into the rectangle for the instance in Figure 2, where circles c_1 and c_6 are first placed in the rectangle to generate an initial configuration, then c_4, c_2, c_3, c_5, c_7 c_8 and c_9 are successively placed according to the MHD rule.

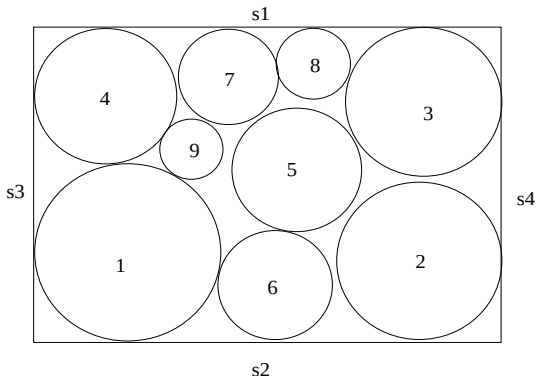

Figure 3: A successful configuration obtained by B1.0 for the instance of 9 circles

```
Procedure B1.0(I)
Begin
   for k := 1 to n − 1 do
      for l := k + 1 to n do
         generate an initial configuration C by placing circles k and l
         into the rectangle;
         generate the corner placement list L;
         if (B1.0Core(C, L) returns a successful configuration)
         then stop with success;
      endfor
   endfor
   stop with failure;
End.

Procedure B1.0Core(C, L)
Begin
   while (there are corner placements in L) do
      for (every corner placement p(c_i, x, y)) do
         calculate its hole degree λ, benefit(p(c_i, x, y)) := λ ;
      endfor
      select the corner placement p(c_i, x, y) with the maximum benefit;
      modify C by placing c_i at (x, y);
      modify L;
   endwhile
   return C;
End.
```

Figure 4: Algorithm B1.0

Algorithm B1.0 is described in Figure 4.

Given a configuration C, we always associate with C the list L of all possible corner placements. Note that L is empty for a successful configuration or a failed one. After placing circle c_i, the list L of corner placements is modified as follows:

- Remove all placements involving c_i;
- Remove all infeasible placements. A placement becomes infeasible because the involved circle would overlap c_i if it was placed;
- Re-calculate the degree λ of the remaining placements;
- If a circle outside the rectangle can be placed inside the rectangle without overlap so that it touches c_i and a circle inside the rectangle or a side of the rectangle, create a new corner placement and put it into L, and compute the degree λ of the new placement.

Algorithm B1.0 was tested on the six instances presented in a later section. Experimental results show that the algorithm is efficient enough to solve some of these instances (see Table 3).

Self look-ahead search strategy and Algorithm B1.5

Given a configuration, B1.0 only focuses on the relations between the circles already inside the rectangle and the circle to be packed. It doesn't examine the relations between the circles outside the rectangle. We apply the *self look-ahead search strategy* proposed by Huang et al. in the works to pack unequal circles into a circular container[10, 11] to solve this problem.

In order to evaluate more globally the benefit of a corner placement and to improve B1.0, we use the result obtained by B1.0 (B1.0Core more precisely) to measure the benefit of a corner placement, which gives our second packing algorithm B1.5 presented below.

At a configuration C, B1.5 examines every corner placement using procedure B1.0Core on a copy of C. It places the circle of the examined placement into the rectangle and then applies B1.0Core to reach a final configuration. If the final configuration is successful, B1.5 stops; otherwise, the density of the final configuration is used to define the benefit of the examined placement, where the configuration density is the ratio between the total surfaces of the circles inside the container and the surface of the rectangle. After examining all placements, B1.5 executes the placement with the maximal benefit to really change configuration C.

Like B1.0, B1.5 tries all pairs of circles before stopping with failure. Figure 5 shows the algorithm B1.5.

Complexity of B1.0 and B1.5

We analyze the complexity of B1.0 and B1.5 in the worst case, i.e., when they cannot find successful configuration, and discuss the real computational cost to find a successful configuration.

```
Procedure B1.5(I)
Begin
   for k := 1 to n − 1 do
      for l := k + 1 to n do
         generate an initial configuration C by placing circle k and l
         into the rectangle;
         generate the corner placement list L;
         if (B1.5Core(C, L) returns a successful configuration)
         then stop with success;
      endfor
   endfor
   stop with failure;
End.

Procedure B1.5Core(C, L)
Begin
   while (there are corner placements in L) do
      for (every corner placement p(c_i, x, y)) do
         let C′ and L′ be copies of C and L;
         modify C′ by placing circle c_i at (x, y);
         modify L′;
         C′ =B1.0Core(C′, L′);
         if (C′ is successful) then return C′;
         else benefit(p(c_i, x, y)) := density(C′);
      endfor
      select the corner placement p(c_i, x, y) with the maximum benefit;
      modify C by placing circle c_i at (x, y);
      modify L;
   endwhile
   return C;
End.
```

Figure 5: Algorithm B1.5

Upper bound of B1.0's complexity

B1.0 is clearly polynomial. Since every pair of circles or sides in the container can give a possible corner placement for a circle outside the container, the length of L is bounded by $O(m^2(n-m))$ if m circles are already placed in the container. For each existing corner placement, d_{min} is calculated using the d_{min} in the last iteration in $O(1)$ time. The creation of new corner placements and the calculus of their degree is also bounded by $O(m^2(n-m))$ since there are at most $O(m(n-m))$ new corner placements (a circle forms a corner position with each side and each circle in the container). So the time complexity of B1.0Core is bounded by $O(n^4)$. B1.0 calls B1.0Core for every pair of circles, its time complexity is bounded by $O(n^6)$.

Upper bound of B1.5's complexity

B1.5 uses a powerful *self look-ahead search strategy* in which the consequence of each possible corner placement is evaluated by applying B1.0Core in full, which allows us to examine the relation between all circles (inside and outside the container). Note that the benefit of a corner placement is measured by the density of a final configuration, which means that we should apply B1.0Core through to the end each time. We are searching for other measures for the benefit of a corner placement so that it is possible to apply B1.0Core only partially without deteriorating solution quality and to reduce run time.

B1.5 uses a $O(n^4)$ procedure to evaluate all $O(m^2(n-m))$ possible corner placements at every iteration. Its complexity is bounded by $O(n^8)$. B1.5 calls B1.5Core for every pair of circles, its complexity is bounded by $O(n^{10})$.

It should be pointed out that the above upper bounds of the complexity of B1.0 and B1.5 are just rough estimations, because most placements are infeasible and removed from L, and the real number of corner placements in a configuration is thus much smaller than the theoretical upper bound $O(m^2(n-m))$. In Table 3, the rows $maxNbCP$ record the maximal number of corner placements in an iteration of B1.0 and B1.5 for the six test instances. These numbers range from 60 to 845 for B1.0, and from 117 to 936 for B1.5 respectively, for the six instances of 20 to 100 circles. The real number of corner placements depends on the number of circles, the diversity of circle radii, and the rectangle width.

The real computational cost of B1.0 and B1.5 to find a successful configuration is still much smaller than the above upper bounds. When a successful configuration is found, B1.0 does not continue to try other pairs of circles, nor B1.5 to exhaust the search space. In fact, every call to B1.0Core in B1.5Core may lead to a successful configuration and stop the execution. Then, the real computational cost of B1.0 and B1.5 essentially depends on the real number of corner placements in a configuration and the distribution of successful configurations. When the rectangle width is not close to the optimal one, there exists many successful configurations, and B1.0 and B1.5 can quickly find such one (see Tables 2 and 6). However, when the rectangle width is very close to the optimal one, few successful configurations exist in the search space, and B1.0 and B1.5 may need to spend more time to find a successful configuration in this case (see Table 3).

Instance	SY1	SY2	SY3	SY4	SY5	SY6
(W, H)*	(17.491, 9.50)	(14.895, 8.50)	(14.930, 9.00)	(24.355, 11.00)	(38.047, 15.00)	(38.647, 19.00)
n	30	20	25	35	100	100
*density**	83.186	81.638	81.940	81.738	82.220	82.243

Table 1: Test instances used in Stoyan and Yaskov's works[7, 8, 9]

	Instance	SY1	SY2	SY3	SY4	SY5	SY6
S.Y.	*(W*, H)*	(17.491, 9.50)	(14.895, 8.50)	(14.930, 9.00)	(24.355, 11.00)	(38.047, 15.00)	(38.647, 19.00)
B1.0	*time(s)*	*F*	<1	<1	2	2	4
B1.5	*time(s)*	186	7	1	2	2	2

Table 2: Run times in seconds to get a successful configuration by B1.0 and B1.5 for the six instances with the given dimension in Table 1; F indicates that no successful configuration is obtained.

Experimental results and analysis

In this section, we evaluate our algorithms B1.0 and B1.5. They are tested on the six instances of Stoyan and Yaskov[7, 8, 9] and compared to the published approaches in the literature. B1.0 and B1.5 are implemented in the C language and all the tests are conducted on a PC under Linux operating system with an Athlon XP2000+ processor and 256 Mo of RAM.

For each of the six instances referred to as $SY1$, ..., $SY6$, Table 1 gives the width and the height of the rectangle, the number of circles to pack into the rectangle, and the corresponding density computed by $100 \times \frac{\sum_{i=1}^{n}(r_i^2 \times \pi)}{W \times H}$. The width and the density for each instance in the table are the best known results in the literature, obtained by the approach of Stoyan and Yaskov[8, 9], denoted by W^* and $density^*$, but they are not optimal. The radii of circles are available from http://www.laria.u-picardie.fr/~hifi accessed 4 June 2004 (the radii of circles for $SY1$ is also provided in Stoyan and Yaskov's work[9]).

First results for the six instances

We directly run B1.0 and B1.5 on the six instances with the given dimension in Table 1, and report the results in Table 2. These results show that

- B1.0 quickly produces a successful configuration for $SY2$, ..., $SY6$, in less than 4 seconds, but it has not found a successful configuration for $SY1$.
- B1.5 outperforms B1.0 and obtains rapidly a successful configuration for all the six instances, within less than 7 seconds for $SY2$, ..., $SY6$, and 186 seconds for $SY1$. The run time for $SY1$ is larger than for the other five instances, which might be explained as follows: the width of $SY1$ is probably closer to the optimal value than the widths in $SY2$, ..., $SY6$.

Figure 6 and Figure 7 illustrate two successful configurations produced by B1.5 for $SY3$ and $SY6$ with the given dimension in Table 1. Their coordinates are available from http://www.laria.u-picardie.fr/~yli accessed 4 June 2004.

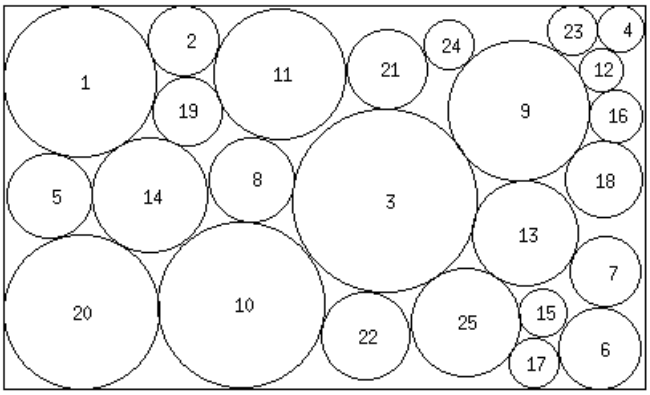

Figure 6: Successful configuration obtained by B1.5 for *SY3* with $W = 14.930$, $H = 9.0$

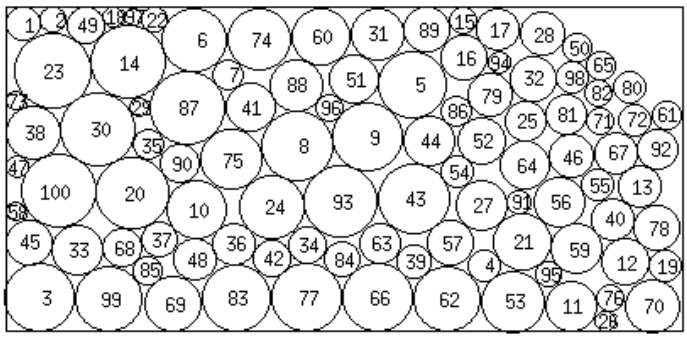

Figure 7: Successful configuration obtained by B1.5 for *SY6* with $W = 38.647$, $H = 19.0$

	Instance	SY1	SY2	SY3	SY4	SY5	SY6
S.Y.	*(W*, H)*	(17.491, 9.5)	(14.895, 8.5)	(14.930, 9.0)	(24.355, 11.0)	(38.047, 15.0)	(38.647, 19.0)
	n	30	20	25	35	100	100
	*density**	83.186	81.638	81.940	81.738	82.220	82.243
B1.0	*W*	17.561	14.735	14.660	23.915	36.547	36.997
	density	82.854	82.525	83.449	83.242	85.595	85.911
	gain	-0.40	**+1.09**	**+1.84**	**+1.84**	**+4.10**	**+4.46**
	time(s)	14	<1	6	20	18052	10704
	maxNbCP	110	60	102	139	845	845
B1.5	*W*	17.291	14.535	14.470	23.555	36.327	36.857
	density	84.148	83.660	84.545	84.514	86.113	86.237
	gain	**+1.16**	**+2.48**	**+3.18**	**+3.40**	**+4.73**	**+4.86**
	time(s)	1628	396	1385	6654	81199	47085
	maxNbCP	202	117	187	246	936	808

Table 3: Minimal widths found by B1.0 and B1.5, gains compared with Stoyan and Yaskov's approach, and run times in seconds used by B1.0 and B1.5 to get a successful configuration under the corresponding minimal width for the six instances

Further results for the six instances

We apply B1.0 and B1.5 to search a width as small as possible for the six instances. First, we apply B1.0 to reach the minimal width, then use B1.5 to improve it. We discuss our results in terms of solution quality and run time.

The minimal width of B1.0 and B1.5 is computed as follows. For each instance, given an upper bound on the width, it is gradually reduced until B1.0 no longer finds a successful configuration. The final upper bound is then taken as the minimal width obtained by B1.0. In the same way, the minimal width of B1.0 taken as a starting point, is gradually reduced until B1.5 no longer finds a successful configuration, and the final upper bound is taken as the minimal width obtained by B1.5.

Table 3 reports the solutions of B1.0 and B1.5 for the six instances. Row (W^*,*H)* and Row $density^*$ are respectively the instance dimension and the corresponding density obtained by the approach of Stoyan and Yaskov, used as a comparison. For B1.0 and B1.5, row W gives the minimal width reached by the algorithm without changing the height. Row $density$ is the density of the successful configuration under the corresponding minimal width. Row $gain$ is the gain over the density obtained by the approach of Stoyan and Yaskov, which is equal to $100 \times \frac{density - density^*}{density^*}$. Row $time(s)$ is the run time in seconds to get a successful configuration under the corresponding minimal width. Row $maxNbCP$ is the maximal number of corner placements in an iteration of B1.0 and B1.5.

These results show that

1. B1.0 improves the best known solutions (those of Stoyan and Yaskov) for $SY2$, ..., $SY6$, with gains from 1.09% to 4.46%. It cannot find a better solution than Stoyan and Yaskov's approach for $SY1$, but the gap is only 0.40%. Note that the improvement is important for the two large instances $SY5$ and $SY6$, with gains of 4.10% and 4.46% respectively. The run time to get a successful configuration with the minimal width is reasonable. Particularly, the run time is small for the first four instances, using less than 20 seconds.

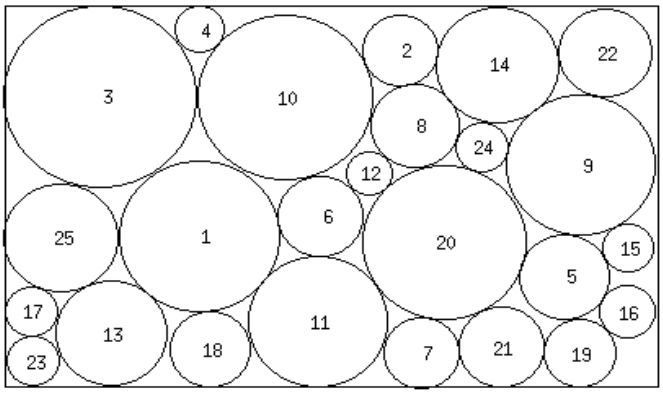

Figure 8: Successful configuration obtained by B1.5 for $SY3$ with $W = 14.470$, $H = 9.0$

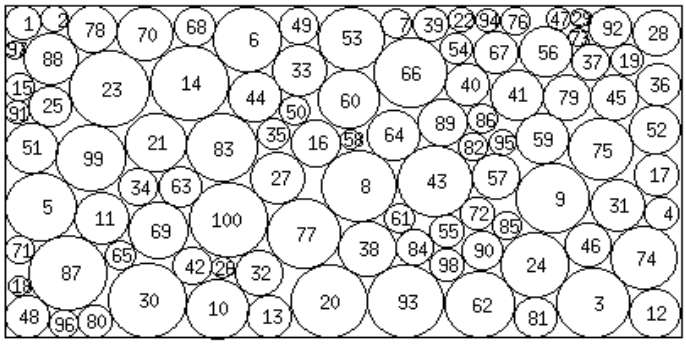

Figure 9: Successful configuration obtained by B1.5 for $SY6$ with $W = 36.857$, $H = 19.0$

2. B1.5 significantly improves the solutions of Stoyan and Yaskov for all six instances. The gains over the solutions of Stoyan and Yaskov respectively are, 1.16% for $SY1$, 2.48% for $SY2$, 3.18% for $SY3$, 3.40% for $SY4$, 4.73% for $SY5$, and 4.86% for $SY6$.

Figure 8 and Figure 9 illustrate two successful configurations produced by B1.5 for $SY3$ and $SY6$ with the minimal width found. The coordinates of circles corresponding to the successful configuration in Figure 8 is given in Table 4, and the coordinates of circles corresponding to other successful configurations with the minimal width found for the six instances are available from http://www.laria.u-picardie.fr/~yli accessed 4 June 2004.

We observed that the computational time of B1.5 to get a successful configuration under the obtained minimal width is considerable. This fact can be explained as follows :

- The minimal widths found by B1.5 are guessed to be very close to the optimal values, even though the optimal ones are not known, which means that B1.5 should search a very large space to get a successful configuration.

i	r_i	x_i	y_i	i	r_i	x_i	y_i	i	r_i	x_i	y_i
3	2.147	-5.0880000	-2.3530000	4	0.549	-2.9079720	-3.9391570	24	0.587	3.3623990	-1.1774226
25	1.273	-5.9620000	0.9534368	13	1.245	-4.8679827	3.2213543	9	1.647	5.5537266	-0.7428660
1	1.782	-2.9072389	0.9152292	17	0.595	-6.6317581	2.6972395	22	1.044	6.1044306	-3.3769135
11	1.547	-0.2748007	2.9530000	21	0.948	3.7825646	3.5520000	5	0.997	5.1894796	1.8759238
7	0.837	2.0010192	3.6630000	23	0.589	-6.5795969	3.8800900	19	0.804	5.5286367	3.6960000
20	1.812	2.5010959	1.0616303	12	0.509	0.8428670	-0.5623519	16	0.627	6.5762021	2.7211319
18	0.904	-2.6420854	3.5881096	8	0.991	1.8624791	-1.6676512	15	0.576	6.6148209	1.2105433
6	0.954	-0.2105970	0.4528242	2	0.833	1.5436216	-3.4635649				
10	1.945	-0.9960284	-2.3377489	14	1.362	3.7119366	-3.1223388				

Table 4: Placement of circles corresponding the successful configuration in Figure 8*;* i is the number of each circle; the order of circles from up to down in each column in the table is the placement order given by B1.5; the configuration in Figure 8 is vertically turned, i.e., the bottom-left corner of the container is moved to the top-left corner due to the used drawing system.

- Our main objective in this paper is to study the behaviour of the proposed heuristics, so we do not make a big effort to optimize our implementation. In spite of this, B1.0 and B1.5 are better than other existing algorithms in terms of solution quality and run time, as is discussed in the following sub-section.

Comparative results with other existing approaches

In addition to Stoyan and Yaskov's approach (noted S.Y.), we also compare our approach with other exsting algorithms in this sub-section. These are three algorithms developed by Hifi et al.: a simulated annealing based Bottom-Left heuristic, denoted by SA[6]; a standard Bottom-Left heuristic with decreasing radius order, denoted by CH[7]; and a genetic algorithm improving CH by searching better radius orders than decreasing radius order, denoted by GA-BH[7]. SA and GA-BH are stochastic algorithms in contrast to B1.0 and B1.5 which are deterministic. CH and GA-BH are also tested on the six instances proposed by Stoyan and Yaskov.

The comparison is made in terms of solution quality and run time. The solution quality is represented by the density of the successful configuration with the minimal width obtained by different approaches. To compare the run times of different approaches, we run B1.0 and B1.5 to get a successful configuration with the minimal widths obtained by other algorithms. It should be pointed out that run times also depend on the type of equipment and implementation. So the time comparison here just gives an indication of the performance of our approach relative to the existing approaches.

Table 5 reports the density gain obtained by each algorithm over the solution of S.Y.. SA, CH and GA-BH did not improve the S.Y. solutions. B1.0 improves the S.Y. solutions for the five instances $SY2$, ..., $SY6$, but not for $SY1$. B1.5 consistently produces better results compared to all the published algorithms for all six instances.

Stoyan and Yaskov did not report the run time of their algorithm for the six instances, except for *SY6* for which about one hour was needed to get the solution in Table 1 on a IBM PC/AT 486[8]. Hifi and M'Hallah[7] reported the run time of CH and GA-BH to solve the six instances.

We run B1.0 and B1.5 to get a successful configuration with the minimal width obtained by CH

	Instance	SY1	SY2	SY3	SY4	SY5	SY6
S.Y.	*density*	83.186	81.638	81.940	81.738	82.220	82.243
SA	*density*	74.357	69.908	65.385	71.796	80.208	79.453
	gain	-10.61	-14.37	-20.20	-12.16	-2.45	-3.39
CH	*density*	79.582	77.535	79.756	80.307	82.220	82.042
	gain	-4.33	-5.03	-2.67	-1.75	0.00	-0.24
GA-BH	*density*	80.960	79.846	81.898	80.549	82.220	82.243
	gain	-2.68	-2.20	-0.05	-1.45	0.00	0.00
B1.0	*density*	82.854	82.525	83.449	83.242	85.595	85.911
	gain	-0.40	**+1.09**	**+1.84**	**+1.84**	**+4.10**	**+4.46**
B1.5	*density*	84.148	83.660	84.545	84.514	86.113	86.237
	gain	**+1.16**	**+2.48**	**+3.18**	**+3.40**	**+4.73**	**+4.86**

Table 5: Density gains obtained by SA, CH, GA-BH, B1.0 and B1.5 over the solutions of Stoyan and Yaskov for the six instances.

for the six instances, and compare the run times of B1.0 and B1.5 with those of CH. Similarly, we run B1.0 and B1.5 to get a successful configuration with the minimal width obtained by GA-BH for the six instances, and compare the run times of B1.0 and B1.5 with the reported average time taken by GA-BH to reach their best solution (recall that GA-BH is stochastic). Run times of CH and GA-BH are measured on a Pentium III 733 MHz and 128 M of RAM, while run times of B1.0 and B1.5 are for an Athlon XP2000+ processor and 256 Mo of RAM, Athlon XP2000+ being about two or three times faster than Pentium III 733. In order to simplify the comparison, the run times of CH and GA-BH are given in Row $time(s)$, and are divided by 3 in Row $time'(s)$. Table 6 reports these results. In addition, to give a global time comparison with the exsiting approaches, we insert the results in Table 2, the run time of B1.0 and B1.5 to get a successful configuration with the minimal width obtained by S.Y. approach.

We observe that, compared to *S*A, CH and GA-BH, B1.0 and B1.5 are all faster, except for $SY5$ for which B1.0 and B1.5 need 2 seconds.

Conclusion and perspectives

We have presented two new greedy algorithms B1.0 and B1.5 for packing unequal circles into a $2D$ rectangular container. The main feature of our approach is the use of the hole degree to evaluate the benefit of a corner placement to obtain B1.0, and the use of B1.0 itself to evaluate more globally the benefit of a corner placement to obtain B1.5. We have compared our approach with the existing ones. Experimental results on the tested instances show that our approach outperforms other approaches in the literature.

We believe that the reason for the effectiveness of B1.0 is the selection for the next circle by the MHD rule, which is inspired from human activity in packing. This rule precisely describes what is a good corner placement for packing unequal circles. The good performance of B1.5 over B1.0 shows that the *self look-ahead search strategy* works very well with a basic heuristic like B1.0.

In the future, we will further study other packing problems.

	Instance	SY1	SY2	SY3	SY4	SY5	SY6
	density	83.186	81.638	81.940	81.738	82.220	82.243
S.Y.	*time(s)*	-	-	-	-	-	-
B1.0	*time(s)*	***F***	**<1**	**<1**	**3**	**2**	**4**
B1.5	*time(s)*	**186**	**7**	**1**	**2**	**2**	**2**
	density	79.582	77.535	79.756	80.307	82.220	82.042
CH	*time(s)*	<1	<1	<1	<1	13	14
CH	*time'(s)*	<1	<1	<1	<1	4	5
B1.0	*time(s)*	**<1**	**<1**	**<1**	**<1**	**2**	**2**
B1.5	*time(s)*	**<1**	**<1**	**<1**	**<1**	**2**	**2**
	density	80.960	79.846	81.898	80.549	82.220	82.243
GA-BH	*time(s)*	25	9	20	15	<1	287
GA-BH	*time'(s)*	8	3	7	5	<1	95
B1.0	*time(s)*	**<1**	**<1**	**<1**	**<1**	**2**	**4**
B1.5	*time(s)*	**<1**	**<1**	**1**	**<1**	**2**	**2**

Table 6: Run time comparison of B1.0 and B1.5 with S.Y., CH, and GA-BH . S.Y. did not report their run time, the corresponding row is marked by '-'. F indicates that no successful configuration is obtained. The time in Row $time'(s)$ is obtained by dividing the value in Row $time(s)$ by 3 . All times are in seconds.

The research in the paper is a continuation of our work and some others in the domain of packing going since 1979[13], and it once again shows that human experience, feeling and thoughts, are rich sources for our mathematical thinking.

Acknowledgement

The authors are thankful to Stoyan, Yaskov, Hifi and M'Hallah for the use of their instances and results, and to two anonymous referees for their helpful comments and suggestions.

References

1 George J.A., George J.M. and Lamer B.W. (1995) Packing different-sized circles into a rectanglar container. *European Journal of Operational Research* 84: 693-712.

2 Lenstra J.K. and Rinnooy Kan A.H.G. (1979) Complexity of packing, covering, and partitioning problems. In: Schrijver A. (ed) *Packing and covering in combinatorics*, Mathematisch Centrum, Amsterdam, pp 275-291.

3 Dowsland K.A. (1991) Palletisation of cylinders in cases. *OR Spektrum* 13: 171-172.

4 Fraser H.J. and George J.A. (1994) Integrated container loading software for pulp and paper industry. *European Journal of Operational Research* 77/3: 466-474.

5 Graham R.L. and Lubachevsky B.D (1996) Repeated patternd of dense packings of equal disks in a square. *The Electronic Journal of Combinatorics* 3, Report No 16.

6 Hifi M., Paschos V. Th. and Zissimopoulos V. A. simulated annealing approach for the circular cutting problem. *European Journal of Operational Research*. In Press, Corrected Proof, Available online 28 September 2003.

7 Hifi M. and M'Hallah R. (2004) Approximate algorithms for constrained circular cutting problems. *Computers and Operations Research* 31: 675-694.

8 Stoyan Yu G. and Yaskov G. (1998) Mathematical model and solution method of optimization problem of placement of rectangles and circles taking into account special constraints. *Int. Trans. Operational Research* 5: 45-57.

9 Stoyan Yu G. and Yoskov G. (2003) A mathematical model and a solution method for the problem of placing various-sized circles into a strip. *European Journal of Operational Research*, In Press, Corrected Proof, Available online 15 May 2003.

10 Huang W. Q., Li Y., Gérard S., Li C.M. and Xu R.C. (2002) A "Learning From Human" Heuristic for Solving Unequal Circle Packing Problem. In Hao J.K. and Liu B.D. (eds) *Proceedings of the First International Workshop on Heuristics*, Beijing, China, pp 39-45.

11 Huang W. Q., Li Y., Jurkowiak B. and Li C.M. (2003) A Two-Level Search Strategy for Packing Unequal Circles into a Circle Container. In Francesca Rossi (ed) *Proceedings of Principles and Practice of Constraint Programming - CP2003*, Kinsale, Ireland. Springer, LNCS 2833, pp 868-872.

12 Huang W. Q. and Moor Sweedler (1992) A practical feasible square packing algorithm for chip manufacture in VLSI. *Technical report 92-20*, The mathematical sciences institute of Cornell University, U.S.A..

13 Huang W.Q. and Zhan S.H. (1982) A quasi-physical method for solving packing problems. *Mathematical Reviews*, American Mathematical Society 82h:52002.