Japan J. Indust. Appl. Math., 21 (2004), 259–278 Area ⟨3⟩

Disk Packing for the Estimation of the Size of a Wire Bundle

Kokichi SUGIHARA[†1], Masayoshi SAWAI[†2], Hiroaki SANO[†3], Deok-Soo KIM[†4] and Donguk KIM[†4]

[†1] *University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan*
[†2] *Yazaki Parts Co., Ltd., 2464-48 Washizu, Kosai-City 431-0431, Japan*
[†3] *Hoct Systems Co., Ltd., 50-5 Takenokaido-cho, Takehana, Yamashina-ku, Kyoto 607-8080, Japan*
[†4] *Hanyang University, 17 Haengdang-Dong, Seongdong-ku, Seoul 133-791, Korea*

Received September 18, 2003
Revised March 22, 2004

A heuristic method for packing disks in a circle is constructed, and is applied to the estimation of the sizes of holes through which given sets of electric wires are to pass. Modern intelligent machines such as planes and cars have a variety of electric systems, and consequently a lot of electric wires run in a complicated way. These wires should pass through holes opened in the walls of the body of a machine. Those wholes should be as small as possible because larger holes weaken the body. The problem of finding the smallest hole is reduced to the problem of finding the smallest circle containing all of given disks without overlap. In the proposed method, a sufficiently large circle is initially constructed, and it is shrunk step by step while keeping all the disks inside. For this purpose a Voronoi diagram for circles is used. Computational experiments show the validity and the efficiency of the method.

Key words: circle packing, circle Voronoi diagram, wire bundling, bundle size, automobile industry

1. Introduction

Modern intelligent cars contain many electric systems. For example, a car has many sensors such as temperature sensors and speed sensors, many controllers such as engine controllers and break controllers, many power electricity such as an engine starter and a power steering, many lights such as head lights and break lamps, and many additional systems such as a global positioning system, a CD player, sound speakers and a radio. Consequently a large number of electric wires run in the body of a car in a complicated manner.

Fig. 1 shows a typical example of a bundle of electric wires used in an actual car. As shown in this example, a set of wires of various sizes are collected together into a cylinder-like bundle, and many bundles are again collected together into a larger bundle. In order to layout those bundles of wires, we have to make holes on walls of the body through which the wires pass. Those holes should be large enough for the bundle of wires to pass, but we do not want to make them unnecessarily large because larger holes will weaken the body.

Hence, we want to estimate the size of a bundle of wires as precisely as possible

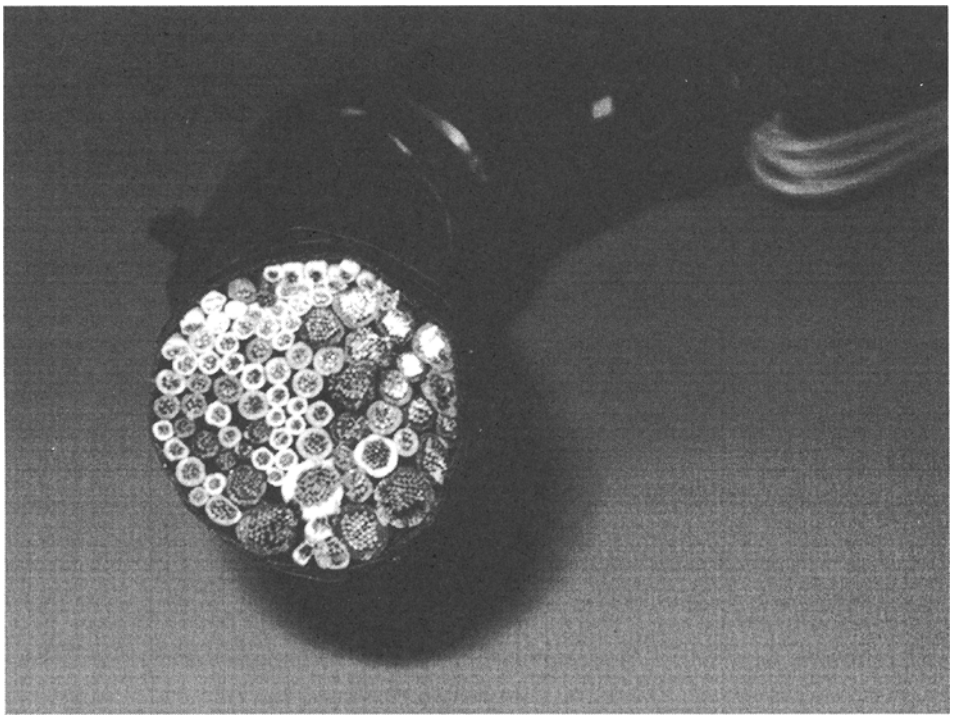


Fig. 1. Example of a bundle of wires used in a car.

in order to determine the sizes of the holes. From the precision point of view the best way is to actually make a bundle using real wires. However we do not want to do that, because we want to know the appropriate sizes of the holes in the design stage, where everything is just on sheets of paper or in computers. Thus, given a set of the sizes of wires, we want to estimate the size of the bundle that would be generated when we put them together tightly. This is the problem we consider in this paper.

Considering the cross section of a wire bundle, we may reduce the problem into a two-dimensional disk packing problem, that is, given a set of disks (corresponding to the cross sections of wires), we want to find the smallest enclosing circle that encloses all the disks without overlap.

Disk packing is an old and hard problem, and has been studied by many mathematicians. Examples of old work can be found in Goldberg [3] and Kravitz [9]. Since these days, various types of packing problems have been studied. They include packing disks in a circle [5, 6, 9, 11, 16], packing disks in a square [4, 13, 14, 20], and packing disks on a sphere [3]. They search for the smallest size of the enclosing figure in a strict sense, and gave solutions only to some restricted numbers of disks. Moreover, they concentrate their attention onto the case where all the disks are of the same size. Actually there is almost no result for disks with different sizes.

Fortunately, however, what we want in order to estimate the size of a wire bundle is not the strictly smallest enclosing circle, but an approximation that is nearly smallest. Such enclosing circles were studied by Drezner and Erkut [2]; they reduced the problem to a nonlinear optimization problem, but they reported that their method was time consuming and could compute the circles enclosing up to 23 disks. What we want, on the other hand, is an enclosing circle that encloses several hundreds of disks with different sizes, and hence we cannot use the previous methods.

In this paper, we propose a new method for finding a small circle that encloses a given set of disks with different sizes. Intuitively speaking, this method simulates a physical process in which we shake the disks inside the enclosing circle while we shrink the enclosing circle step by step until the disks cannot move any more.

The structure of the paper is as follows. In Section 2, we describe the method currently used in industry and discuss its insufficiency. In Section 3, we give a basic structure of our algorithm, which simulates a shrink-and-shake procedure. In Section 4, we accelerate our algorithm using generalized Voronoi diagrams. In Section 5, we give some experimental results to show the behavior and the performance of our algorithm. Finally we give some concluding remarks in Section 6.

2. Bundle Coefficient Approach

Let us denote by c_i a disk with radius r_i. The location of c_i is not fixed; we can move c_i in the plane freely. Suppose that we are given a set $C = \{c_1, c_2, \ldots, c_n\}$ of n such disks. Our goal is to find a nearly smallest circle that encloses all the disks without overlap in reasonably small computation time.

Table 1 shows an example of a disk set that actually arises in an existing car. In this table, the left column shows the category names of wires, the middle column

Table 1. Wires belonging to a bundle in an actual car.

category of a wire	radius (mm)	number
23A	1.8	3
20G	0.7	11
31G	0.8	31
44G	0.9	5
45G	1.05	9
51G	1.3	12
25J	0.55	25
28J	0.65	54
30J	0.75	1
31J	0.9	6
22X	0.9	4
10E	1.75	1
total		162

shows the radii of the perpendicular sections of the wires, and the right column shows the number of wires belonging to each category and each size. Hence, in this example, the bundle contains 162 wires, among which the largest one is more than three times larger in radius than the smallest one.

We first describe what is currently being done in industry. They have a secret number, say B, called a "bundle coefficient". Let S be the sum of the areas of the disks:

$$S = \sum_{i=1}^{n} \pi {r_i}^2.$$

Then, they estimate the diameter of the bundle by $B\sqrt{S}$.

This estimation is based on the assumption that in the section of a wire bundle, the ratio of the area occupied by the wires to the empty area is constant.

However, this is not true in general. A typical example is shown in Fig. 2. Fig. 2 (a) shows the case where all the disks are equal in size, and they are packed tightly in the way just like a nest of bees. In Fig. 2 (b), on the other hand, additional smaller disks are inserted in the empty space. Therefore, the ratio of the occupied area is larger in Fig. 2 (b) than in Fig. 2 (a). This example implies that the bundle-coefficient method does not work well, because the distribution of wire sizes changes from bundle to bundle.

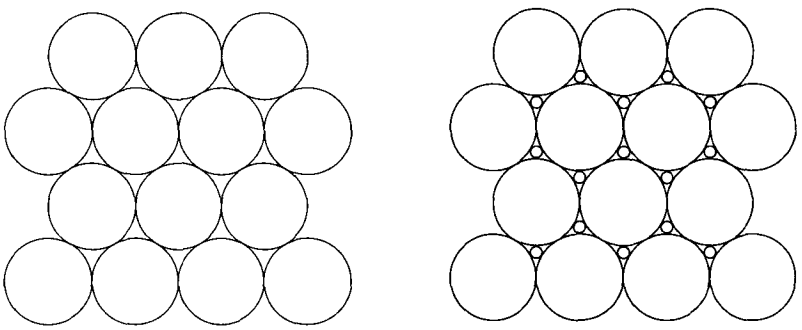


Fig. 2. Two packings in which the ratio of the empty space is different.

Actually there are more than one bundle coefficient in industry, and experienced persons select according to their inspiration the one that seems the most suitable for each set of wires. This kind of estimation cannot guarantee the quality of the result, and hence a more logical method is required.

3. Shrink-and-Shake Algorithm

In this section we present the basic structure of the proposed algorithm. First, we show an example of the behavior of our algorithm, next sketch the basic structure, and then describe how to simulate shaking disks in a circle.

3.1. Rough illustration of the behavior

Fig. 3 illustrates the behavior of the algorithm which we are proposing. As shown in (a), we first place a given set of disks without overlap and construct an enclosing circle. Next, we generate another circle that is a little smaller than the current enclosing circle, as shown in (b). We call this new circle the *target circle.* At this stage, there are some disks that are partly protruding the target circle.

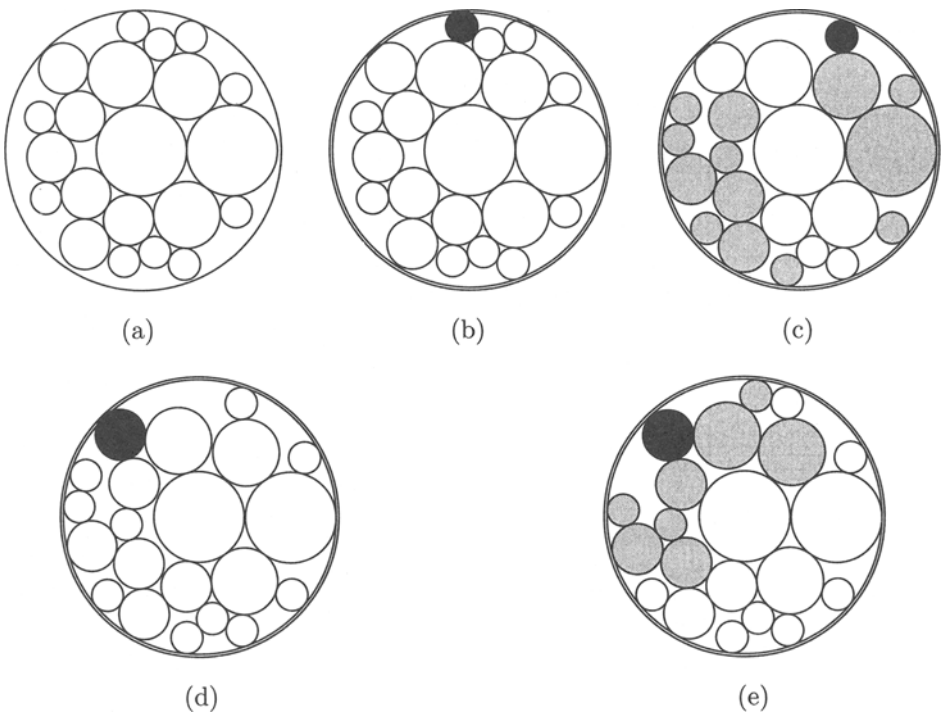


Fig. 3. Shrink-and-shake strategy.

So we next try to push such protruding disks toward inside the target circle. For this purpose we choose one protruding disk. Suppose that we now choose the dark gray disk in (b). Then, instead of pushing this disk directly, we try to move other disks as far as possible from the dark disk, just as we shake the disks; at this point we do not shake much, but we give just one stroke of shaking. Fig. 3 (c) shows the result of a stroke of shaking, where light gray disks are those that were moved by the shaking. Note that as the result of this shaking, the dark disk was moved inside the target circle, and that at the same time some other disks that were previously protruding the target circle were also moved inside.

There still remain some protruding circles, and hence we choose one shown by the dark disk in (d). We give another stroke of shaking in such a way that all the movable disks were moved toward opposite to the dark disk. Then, we get the next configuration as shown in (e).

We repeat a similar procedure until either all the protruding disks are moved inside the target circle or the shaking procedure does not work any more. In the former case we succeed in shrinking, and hence we update the current enclosing circle and generate a still smaller target circle. In the latter case, on the other hand, we fail in shrinking, and hence we generate a new target circle between the current enclosing circle and the current target circle. In both cases we repeat the whole shaking procedure. This is a kind of binary search in the sense that we adjust the size of the target circle according to whether we succeed in pushing the protruding disks or not.

3.2. Basic structure of the algorithm

Now we describe the basic structure of our algorithm. In the next algorithm we use two small positive parameters p and ε. The parameter p is used for determining the radius of the target circle in such a way that the radius of the target circle is $(1-p)$ times the radius of the current enclosing circle; p is halfened in the processing so as to realize the binary search. The parameter ε, on the other hand, is used for judging the termination of the procedure in such a way that the algorithm terminates when the difference of the radii of the enclosing circle and the target circle becomes smaller than ε; ε is fixed throughout the procedure. Square brackets in the procedure represent additional description for the readers.

Algorithm 1 (basic algorithm).
Input: n disks c_i with radius r_i, $i = 1, 2, \ldots, n$, positive number p that is small enough (for example, $p = 0.02$), and a sufficiently small positive number ε (for example, $\varepsilon = \min(r_1, r_2, \ldots, r_n)/1000$).
Output: Non-overlapping placement of the n disks and their enclosing circle with radius R such that R is nearly smallest.
Procedure:

1. [Initialization] Place $c_1, c_2, \ldots, c_n$ without overlap, and find their enclosing circle D. Let the radius of D be R. $R' \leftarrow (1-p)R$.
2. Choose the target circle D' whose center is the same as that of D and whose radius is R'.
3. [Main repetition] While there are protruding disks, choose one, say c_i, and do the following.
   3.1. Sort the remaining disks in the decreasing order of the distances from c_i.
   3.2. Choose the remaining disks one by one in this order, and if it can be moved to some place in D' which is farther than the present location from c_i, move it to the farthest possible location.
   3.3. If there is not enough space to push c_i into D', go to Step 5. Otherwise, push c_i into D'.
4. [Succeed in shrinking] $D \leftarrow D'$, $R' \leftarrow (1-p)R'$ and go to Step 2.
5. [Fail in shrinking] $p \leftarrow p/2$ and $R' \leftarrow (1-p)R$. If $R - R' > \varepsilon$ go to Step 3. Otherwise, report the present enclosing circle D and the associated placement of $c_1, c_2, \ldots, c_n$, and stop. ■

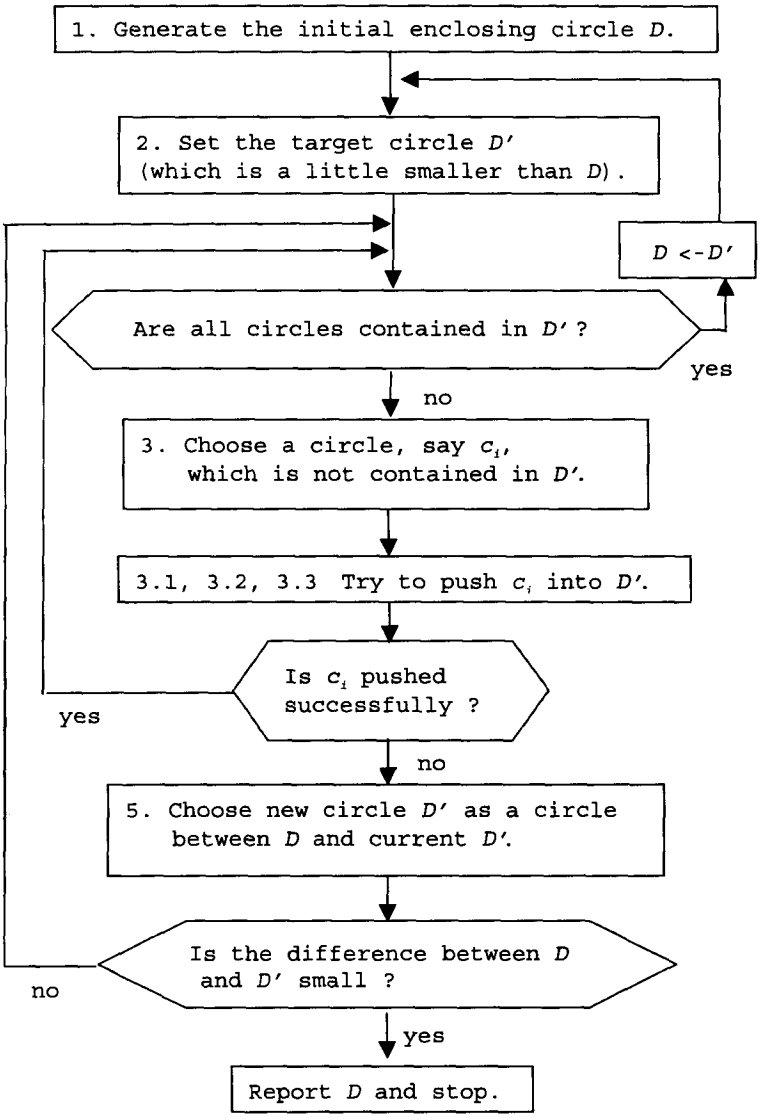


Fig. 4. Flow chart of Algorithm 1.

Fig. 4 shows the flow chart of Algorithm 1. The numbers in this figure represent the step numbers of the procedure of Algorithm 1.

3.3. How to push protruding disks

The most important part of Algorithm 1 is Step 3, which tries to push the protruding disks into the target circle. Step 3.1 of Algorithm 1 can be done in $O(n \log n)$ time by a standard sorting technique such as the heap sort or the merge sort. On the other hand, Step 3.2 and Step 3.3 are not trivial, and take much time

if they are executed naively.

Suppose that we choose disk c_j at Step 3.2 and want to move it farther from the protruding disk c_i. Since the target circle D' is only a little smaller than the current enclosing circle, we can expect that only a small part of the protruding disk c_i is outside D'. Hence, let us assume that the center of the protruding disk c_i is inside D'. Then, at the farthest possible location of the chosen disk c_j, c_j touches two other disks, or touches one disk and the target circle.

Keeping this fact in mind, we can describe Step 3 in more detail. Let $S = \{c_1, c_2, \ldots, c_n, D'\}$ be the set of the n given disks and the target circle D' in their current locations. For two sets X and Y, let $X \backslash Y$ denote the set of elements belonging to X but not belonging to Y. Then, Step 3 of Algorithm 1 can be executed by the next algorithm.

Algorithm 2 (naive realization of Step 3 of Algorithm 1).

3. While there are protruding disks, choose one, say c_i, and do the following.
   - 3.1. Sort the remaining disks in the decreasing order of the distances from c_i.
   - 3.2. Choose the remaining disks one by one in this order and do the following 3.2.1, 3.2.2 and 3.2.3.

     Comment: Let the chosen disk be $c_j \in S \backslash \{D', c_i\}$.
     - 3.2.1. Set storage T empty.
     - 3.2.2. For every pair of two elements $c_k, c_l \in S \backslash \{c_i, c_j\}$, do the following.
       - (1) Compute the new location of the disk c_j touching c_k and c_l simultaneously, where if c_k or c_l coincides with D', c_j touches D' from inside.
       - (2) If there is no such location, go to (4).
       - (3) For each c_j' of such locations, if c_j' is farther from the protruding disk c_i than c_j, check whether c_j' is inside D' and c_j' does not intersect with any disk in $S \backslash \{c_j, D'\}$ and if the result is true, insert c_j' into T.
       - (4) Go to 3.2.2 to check the next pair of circles.
     - 3.2.3. If T is not empty, move c_j to the location in T that is the farthest from c_i.
   - 3.3. If there is no space to push c_i into D', go to Step 5 of Algorithm 1. Otherwise, push c_i into D' (and go to Step 4 of Algorithm 1). ■

Step 3.2 of Algorithm 2 requires $O(n^4)$ time, because for each triple of c_j, c_k and c_l, we have to check the intersection of c_j' at the new location against all the other disks at Step 3.2.2 (3). Step 3.2 is repeated many times in Algorithm 1, and hence the time complexity of Algorithm 1 is very large.

In order to avoid such high computational cost, we next consider how to decrease the time complexity.

4. Acceleration of the Algorithm

To decrease the time complexity of Algorithm 2, we employ the circle Voronoi diagram.

Let $c_1, c_2, \ldots, c_n$ be non-overlapping disks and D be an enclosing circle. The

region inside D and outside $c_1, c_2, \ldots, c_n$ can be partitioned into $n+1$ subregions and their boundaries in such a way that the points nearer to D than to any disks constitute the subregion associated with D and all the points nearer to c_i than to any other disks or D constitute the subregion associated with c_i for $i = 1, 2, \ldots, n$. This partition is called the *circle Voronoi diagram* inside D for disks $c_1, c_2, \ldots, c_n$ [15]. An example of the circle Voronoi diagram is shown in Fig. 5. The circle Voronoi diagram can be constructed in $O(n \log n)$ time by the plane sweep method [21] or the divide-and-conquer method [10, 17]. From an average complexity point of view, the circle Voronoi diagram can also be constructed efficiently from the point Voronoi diagram or from the Laguerre Voronoi diagram by flip operations [7, 8].

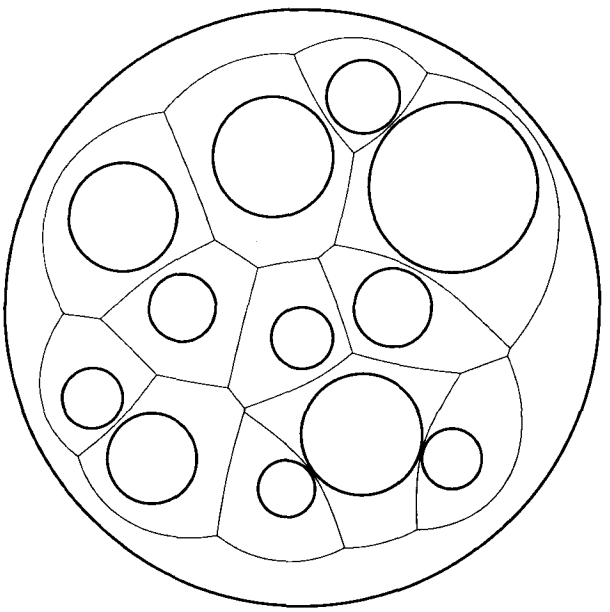


Fig. 5. Circle Voronoi diagram inside an enclosing circle.

Consider Step 3.2 of Algorithm 2 again, where we want to move the disk c_j to the location farthest from the protruding disk c_i. Let V be the circle Voronoi diagram inside D' for disks $c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_n$. A point on an edge of the circle Voronoi diagram is in equal distance from the both side disks, say c_k and c_l, and the other disks are farther. Hence, a new disk c'_j that touches two disks c_k and c_l and that does not intersect with other disks has its center on the edge shared by the subregions associated with the two touching disks c_k and c_l. Therefore, the candidates of the new location of c'_j can touch only such pairs of disks whose subregions share an edge of the circle Voronoi diagram V. The number of edges of V is of $O(n)$ [15], and hence the number of pairs c_k and c_l that should be checked

in Step 3.2.2 of Algorithm 2 can be reduced from $O(n^2)$ to $O(n)$.

Moreover, in Step 3.2.2 (3) we check whether the new disk c_j' intersects with other disks, which requires $O(n)$ time. However, this check can be replaced by the check whether or not the center of c_j' is on the associated edge of V. Hence, the time complexity of Step 3.2.2 (3) decreases from $O(n)$ to $O(1)$.

Summarizing the above consideration, we can construct the next algorithm which replaces Algorithm 2.

Algorithm 3 (improved version of Algorithm 2).

3. While there are protruding disks, choose one, say c_i, and do the following.
   - 3.1. Sort the remaining disks in the decreasing order of the distances from c_i.
   - 3.2. Choose the remaining disks one by one in this order, and do the following 3.2.1, ..., 3.2.4.

     Comment: Let the chosen disk be $c_j \in S \backslash \{D', c_i\}$.
     - 3.2.1. Set storage T empty.
     - 3.2.2. Construct the circle Voronoi diagram V inside D' for disks $c_1, \ldots, c_{j-1}$, $c_{j+1}, \ldots, c_n$.
     - 3.2.3. For each edge of V, find the associated two side disks, say c_k and c_l, check whether the disk $c_j{}'$ with radius r_j touching c_k and c_l has its center on the edge, and if the answer is "yes", insert the disk c_j' into T.
     - 3.2.4. If T is not empty, move c_j to the location in T that is the farthest from c_i.
   - 3.3. If there is no space to push c_i into D', go to Step 5 of Algorithm 1. Otherwise, push c_i into D' (and go to Step 4 of Algorithm 1). ■

Recall that Step 3.2 of Algorithm 2 requires $O(n^4)$ time. On the other hand, Step 3.2 of Algorithm 3 requires only $O(n^2 \log n)$ time because of the following reason. Step 3.2.1 and Step 3.2.4 can be done in $O(1)$ time. Step 3.2.2 can be done in $O(n \log n)$ time in the worst case [10, 21]. Step 3.2.3 requires in $O(n)$ time because the circle Voronoi diagram for n circles has only $O(n)$ edges. In Step 3.2, Steps $3.1 \sim 3.4$ are repeated $O(n)$ times, and hence the total time complexity of Step 3.2 is $O(n^2 \log n)$.

We can further reduce the time complexity of Step 3.2 in the following way. In this algorithm Step 3.2.2 is repeated $n-1$ times, but the circle Voronoi diagrams constructed in this step are similar to each other. Therefore, instead of constructing these Voronoi diagrams independently, we can modify one diagram to get another. Suppose we already have the circle Voronoi diagram for $n-1$ disks $c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_n$. Then, we get the circle Voronoi diagram for $n-1$ disks $c_1, \ldots, c_j, c_{j+2}, \ldots, c_n$ first by deleting c_{j+1} and next by inserting c_j. The deletion and insertion of one disk requires only local change of the diagrams, and this change requires $O(1)$ time on the average [15]. Hence, we can expect that Step 3.2 requires $O(n^2)$ time on the average.

5. Experimental Results

In this section we presents actual behavior of the proposed algorithm from both output quality point of view and the time complexity point of view.

5.1. Example of the behavior

Fig. 6 shows an example of the behavior of our algorithm for the set of disks given in Table 1.

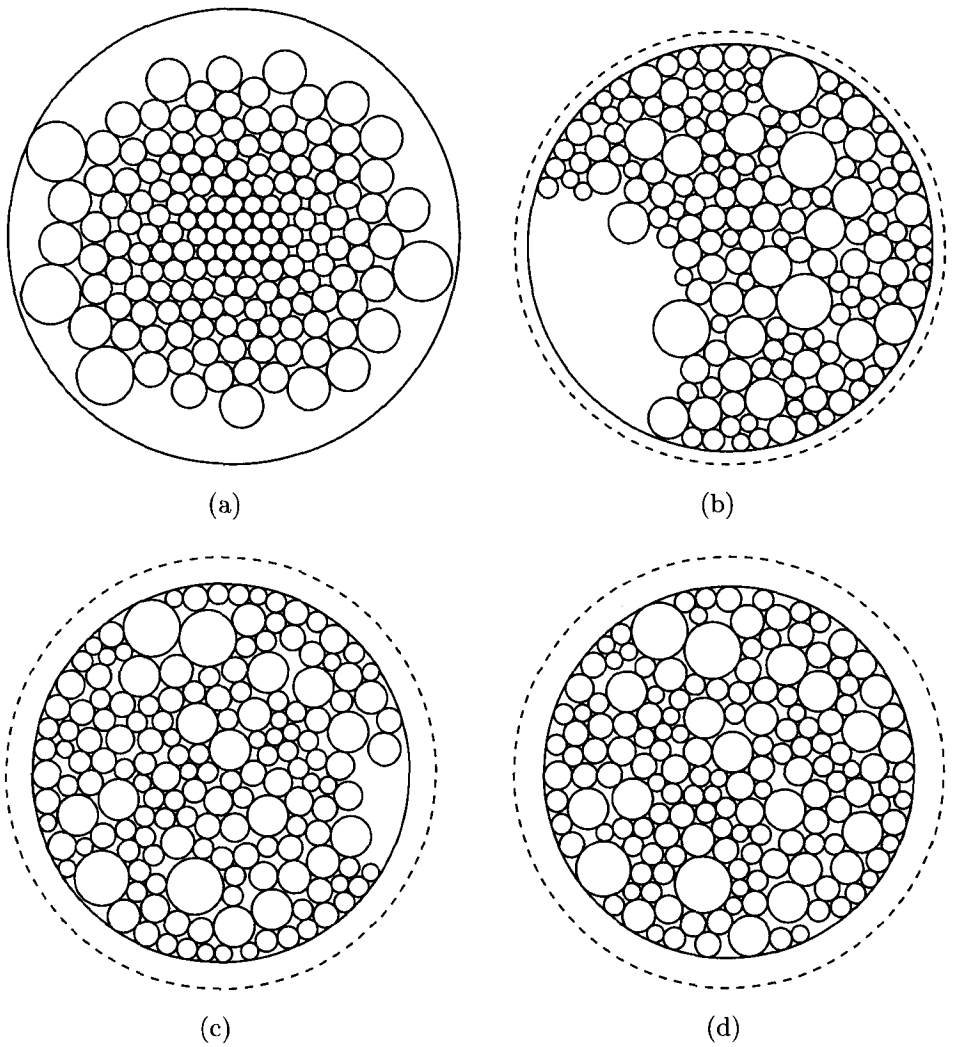


Fig. 6. Behavior of the algorithm for the set of disks in Table 1.

Fig. 6 (a) shows an initial placement of disks and an initial enclosing circle. To get this initial enclosing circle, we first shuffled the disks and fixed a linear order, next place the center of the first disk at the origin of the coordinate system, then

placed the other disks one by one in this order in the position nearest to the origin, and finally generate the smallest enclosing circle with the center at the origin.

Fig. 6 (b) and (c) show the placements obtained at the end of the third repetition and 7th repetition, respectively, of Step 3 of Algorithm 1, where the parameter values $p = 0.02$ and $\varepsilon = (\min_i r_i)/1000$ were used. Fig. 6 (d) shows the final result.

From this figure, we can understand that our procedure is in a sense similar to shaking the disks while shrinking the enclosing circle.

5.2. Quality of the enclosing circle

The output of Algorithm 1 depends on the initial placement of the disks. For the set of disks in Table 1, we chose 10 different linear orders of the disks at random, and observed the outputs of Algorithm 1. The result is shown in Table 2.

Table 2. Radius of the enclosing circles obtained by Algorithm 1.

trial number	no. of target circles	no. of successes	no. of failures	radius [mm]	total area / disk area
1	40	11	29	11.73	1.236
2	15	6	9	11.86	1.264
3	21	10	11	11.78	1.246
4	11	3	8	11.88	1.268
5	18	9	9	11.87	1.266
6	31	12	19	11.77	1.244
7	33	12	21	11.76	1.242
8	31	12	19	11.74	1.239
9	21	6	15	11.77	1.244
10	24	9	15	11.81	1.253
average	24.5	9.0	15.5	11.796	1.2502
variance	81.8	9.56	44.3	0.0030	0.00014

The leftmost column in this table represents a sequential number of trials. The second column represents the number of target circles set in Step 2; this number corresponds to the number of repeats of the main step, Step 3, of Algorithm 1. The third and the fourth columns represent the number of the success cases and that of the failure cases in shrinking the enclosing circle. Hence, the numbers in the second column equal to the sums of the numbers in the third and the fourth columns. The fifth column represents the computed radius of the enclosing circle. The rightmost column represents the ratio of the area of the enclosing circle to the sum of the areas of the disks; the square root of this value corresponds to the secret number used in industry.

From this table we can see that the number of the repeats of Step 3 varies very much, while the variance of the computed radius of the enclosing circles is very small. The main reason for large variance of the member of repeats is that the success of shrinking with large p much depends on the initial placement of the

disks. On the other hand, the main reason for small variance of the computed radius is that the success of shrinking with small p does not depend much on the initial placement. Thus, our algorithm can compute the radius in a stable manner. Hence, we can expect that the algorithm will give a good estimate of the size of the enclosing circle.

Table 3 shows the actual sizes of the wire bundles together with the computed sizes. In this table, we represent the data about 20 wire bundles, which are numbered as in the leftmost column. The second column represents the number of wires in each bundle, and the third and the fourth columns represent the diameter of the smallest wire and that of the largest wire. The fifth column represents the measured diameters of the actual wire bundles, while the sixth column represents the computed diameters. The seventh column represents the errors, i.e., the differences between the computed diameters and the actual diameters. The rightmost column represents the relative errors, i.e., the ratio of the absolute value of the error to the measured diameter of the actual bundle of wires.

Table 3. Comparison of actually measured diameters and computer-estimated diameters of wire bundles.

bundle number	no. of wires	smallest wire [mm]	largest wire [mm]	measured diameter [mm]	computed diameter [mm]	error [mm]	relative error [%]
1	59	1.4	8.0	19.2	19.2	0.0	0.00
2	68	1.4	8.0	20.8	21.0	0.2	0.96
3	62	1.4	8.0	20.1	20.0	−0.1	0.50
4	73	1.4	8.0	21.7	22.0	0.3	1.38
5	68	1.4	8.0	21.2	21.2	0.0	0.00
6	79	1.4	8.0	22.7	22.9	0.2	0.88
7	71	1.4	8.0	22.1	21.8	−0.3	1.36
8	82	1.4	8.0	23.5	23.7	0.2	0.85
9	43	1.4	8.0	18.1	18.0	−0.1	0.55
10	43	1.4	8.0	17.3	17.4	0.1	0.58
11	77	1.6	5.5	21.6	21.1	−0.5	2.31
12	93	1.6	5.5	22.9	22.6	−0.3	1.31
13	60	1.6	5.5	18.6	18.1	−0.5	2.69
14	83	1.6	5.5	22.0	21.7	−0.3	1.36
15	101	1.6	5.5	23.5	23.2	−0.3	1.28
16	107	1.6	5.5	24.0	23.8	−0.2	0.83
17	97	1.6	5.5	22.9	22.6	−0.3	1.31
18	59	1.6	5.5	18.5	18.0	−0.5	2.70
19	72	1.6	5.5	19.7	19.4	−0.3	1.52
20	83	1.6	5.5	22.0	21.7	−0.3	1.36

From this table, we can see that the computed diameters are very close to the actual size of the wire bundles. Indeed, they were much closer than we had expected. This closeness might be understood in the following way.

There are some factors that might cause the computed diameter smaller than the actual size.

First, the actual electric wires are covered by insulator, and the insulator usually has large friction. Our algorithm, on the other hand, implicitly assumes that the wires have no friction. Because of this assumption, disks can move in our algorithm even if free space is very narrow, which might make the enclosing circle smaller than an actual bundle.

Secondly, our algorithm solves a two-dimensional packing problem while the actual bundle of wires has a three-dimensional structure. If the wires were strictly straight, the two-dimensional disk packing problem might be a good approximation of the bundle. However, actual wires are curved, and form complicated three-dimensional structure. Consequently, there might be larger empty space in an actual bundle of wires than in the output of our algorithm. This fact might also make the enclosing circle smaller than an actual bundle.

On the other hand, there is a factor that might cause the computed diameter larger than the actual size. In our formulation, we assumed that the section of a wire, i.e., a disk, does not change its shape. However, the insulator covering a wire is not strictly rigid; it is slightly flexible. Hence, when we hold a bundle of wires in our hand tightly, the insulation deforms in such a way that part of empty space is occupied by the deformed insulator. This phenomenon might make the actual size of the wire bundle smaller.

In this way there are both factors that might make the bundle size larger and that might make it smaller. These factors seem to cancel each other, and result in the closeness of the actual size and the computed size of the wire bundle. This is our current understanding of the data in Table 3.

5.3. Actual computational complexity

In order to evaluate the effect of acceleration using circle Voronoi diagrams, we measured the computation times of Algorithm 1 implemented together with Algorithm 2 (the naive method) and that with Algorithm 3 (the improved method). In this experiment, we used n disks of the same radius r as the input for various values of n.

Since the disks are of the same size, the associated Voronoi diagram coincides with the ordinary Voronoi diagram of the centers of the disks. So we used our fast and robust software VORONOI2, which was based on the topology-oriented principle and has the $O(n)$ average time complexity for uniformly distributed generators [18, 19].

Both of the algorithms were implemented in FORTRAN, compiled by G77 compiler in Solaris Operating System, and executed by ULTRA10 Work Station of Sun Microsystems.

Our primal goal in this experiment is to compare the CPU time of Algorithm 2

and Algorithm 3 for various n, the number of disks. For this purpose we first tried to observe the behavior of these algorithms in our whole computational procedure given by Algorithm 1. However, we found this was not suitable for our purpose. This situation can be understood when we see Fig. 7.

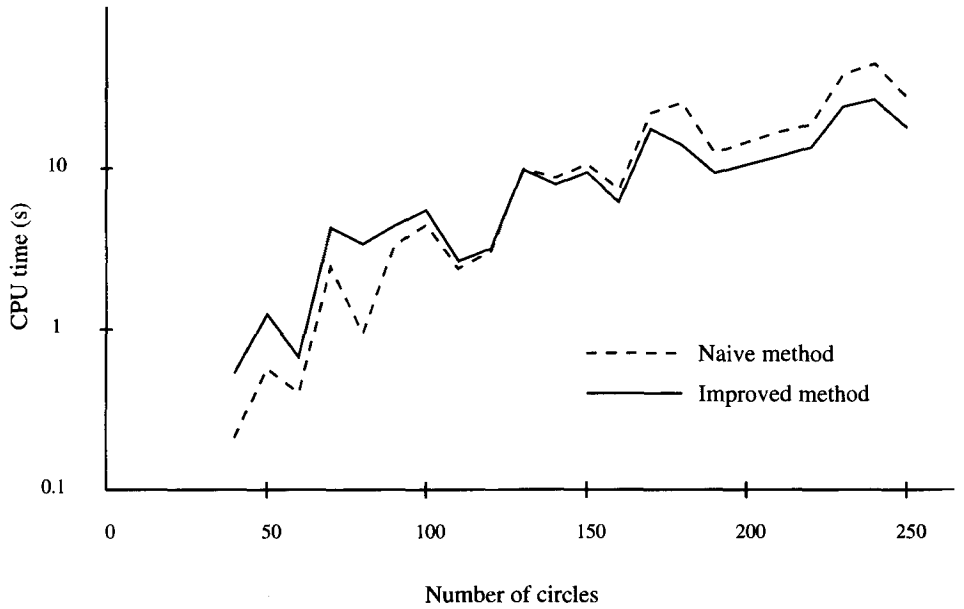
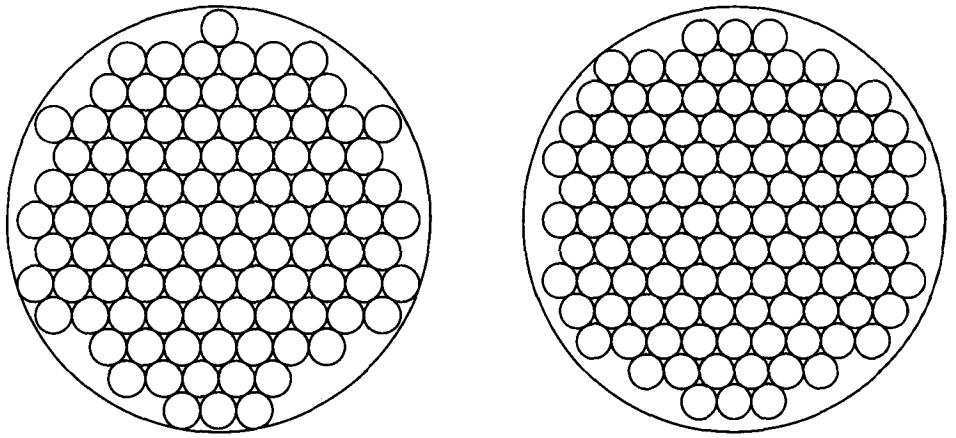


Fig. 7. CPU times of the naive method and the improved method for natural initial placements.

Fig. 7 shows the CPU times of Algorithm 2 (broken line) and Algorithm 3 (solid line) for $n = 40, 50, \ldots, 250$. The horizontal axis represents the number n of disks in linear scale and the vertical axis represents the CPU time in logarithmic scale. For each value of n, we gave the initial placement of the n disks (computed in Step 1 of Algorithm 1) to Algorithms 2 and 3, executed the algorithms ten times and took the average. Hence, each plotted point in this figure shows the CPU time for executing one stroke of the shrink-and-shake procedure. We can see that the observed CPU times behave unstably, and even the monotonicity is not satisfied.

This kind of instability comes from the nature of the problem. For example, both algorithms ran faster for $n = 110$ than for $n = 100$. We understand the reason for this phenomenon when we see the initial placements of disks shown in Fig. 8.

In Step 1 of Algorithm 1, the disks are placed one by one without overlap at the location nearest to the origin in such a way that a new disk touches two of the old disks, and the initial enclosing circle is chosen as the smallest one centered at the origin. Fig. 8 shows the initial placement of n disks and the enclosing circle; (a) shows the case for $n = 100$ while (b) for $n = 110$. For $n = 110$, the initial

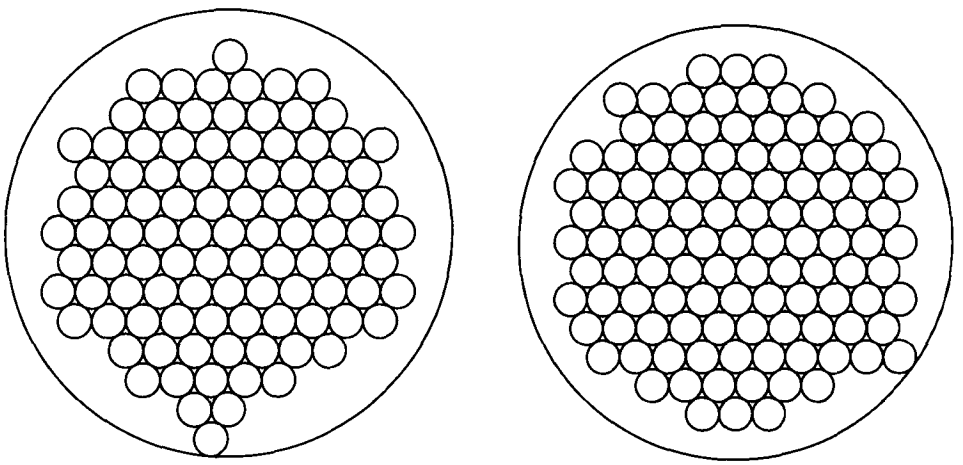


(a) (b)

Fig. 8. Natural initial placement: (a) $n = 100$; (b) $n = 110$.

enclosing circle touches only one disk, and hence the shrink-and-shake procedure requires to push only one protruding disk. For $n = 100$, on the other hand, the initial enclosing circle touches many disks, and hence many protruding disks arises when the enclosing circle is shrunk, and the algorithms should push all of them. This is the reason why the 100 disks need more CPU time than the 110 disks.

Thus the unstable behavior of the CPU times in Fig. 7 comes from the fact that the number of protruding disks varies in a complicated way when we increase n.



(a) (b)

Fig. 9. Artificial initial placement: (a) $n = 100$; (b) $n = 110$.

Hence, in order to avoid this kind of complication, we used a more artificial initial placement as shown in Fig. 9. For a given n, we placed the first $n-1$ disks in the way described above, while we placed the nth disk in the possible farthest location while touching two of the old disks. Fig. 9 (a) and (b) show the resulting placements for $n = 100$ and $n = 110$, respectively. By this convention, we can observe the behavior of Algorithms 2 and 3 for the case where there is only one protruding disk.

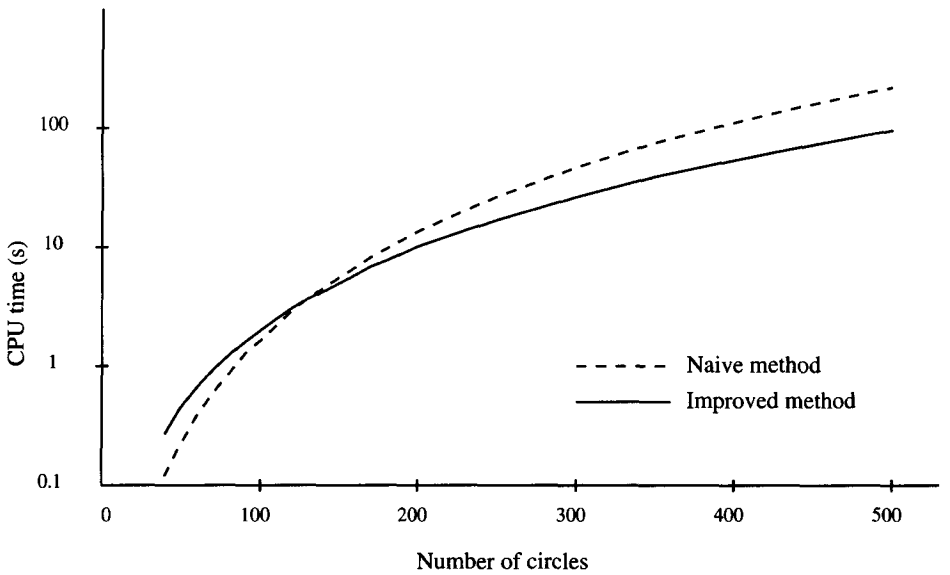


Fig. 10. CPU times of the naive method and the improved method for the input with exactly one protruding disk.

Fig. 10 shows the result of this experiment. The horizontal axis represents the number n of disks in linear scale, and the vertical axis represents the CPU time in logarithmic scale. The broken line shows the CPU times of the naive method (Algorithm 2) while the solid line shows those of the improved method (Algorithm 3).

From this figure we can see that the naive method runs faster for $n \leq 130$, while the improved method shows better performance for $n \geq 140$. Note that the input disks are of the same size in this experiment, and consequently the circle Voronoi diagram coincides with the ordinary Voronoi diagram of the centers of the disks. Therefore, the Voronoi diagram could be constructed faster than the general case where the disk sizes are not necessarily the same. Hence, in the general case where the disks are of different sizes, the improved method will run faster than the naive method only for n much larger than 140.

On the other hand, the wire bundles arising in automobile industry usually have 50 to 200 wires. Hence, our conclusion is that for the purpose of estimating

the sizes of wire bundles in automobile industry, we should use the naive method.

We should add some remarks to this conclusion. The time complexity of $O(n^4)$ of Algorithm 2 comes from a very pessimistic evaluation. In actual implementation of the software used in our experiment, we tuned up the procedures of the naive method. For example, when we got $O(n^4)$ time complexity, we assumed that Step 3.2.2 (3) is repeated $O(n)$ time because $O(n)$ candidates of new locations for c_j are generated in Step 3.2.2 (1). In the actual implementation, on the other hand, the number of the candidates of locations for c_j is much smaller. Indeed, if a candidate of a location is nearer to the protruding disk c_i than the farthest location found by that time, we need not do Step 3.2.2 (3) for this candidate. Moreover, for many pairs of c_k and c_l, there is no candidate location at all for c_j touching both of them, and hence we need not do Step 3.2.2 (3) for those pairs. Thus, the actual CPU time was much smaller than the theoretical worst-case complexity $O(n^4)$. This is part of the reason why the naive method is not too slow, and can be used for packing problems with moderate sizes found in industry.

6. Concluding Remarks

In this paper, we reduced the problem of estimating the size of a wire bundle to the two-dimensional disk packing problem, and proposed a shrink-and-shake method for solving this problem (Algorithm 1). The proposed method is heuristic in its nature, but the experiments show that the output is stable in the sense that the variance of the estimated size of the enclosing circle is very small. Moreover, the size of the enclosing circle obtained by our algorithm is close to the actual size of the wire bundle; it is actually close enough for our original goal of estimating the size of the hole we should make.

From the time complexity point of view, we presented two methods, a naive method (Algorithm 2) and an improved method (Algorithm 3), for the main part of the shrink-and-shake procedure. Experiments show that the improved method gives better performance for large numbers of disks. Also we found that the naive method is useful to solve moderate sizes of problems arising in actual industry.

Our next question is how large the number of disks for the improved method to show a better performance than the naive method in the case where the sizes of disks are different. This is one of our future problems.

There are many related problems for future. In order to make a better estimation of the size of a wire bundle, we need to consider the friction of insulator and the three-dimensional structure of wires.

In this paper we presented the shrink-and-shake strategy, but other strategies might be also possible. For example, when we push the protruding disk toward inside the enclosing circle, the force propagates from disk to disk. This physical phenomenon can also be formulated in a mathematical manner in order to get another strategy.

As for the disk packing problem, we can consider many variants. For example, other problems are obtained when we replace the enclosing circle with other shapes,

such as an enclosing square, an enclosing rectangle, an enclosing triangle and an enclosing ellipse. It might be also a challenge to consider a non-convex enclosing shape.

Acknowledgments. The first author was supported by the 21st Century COE Program on Information Science and Technology Strategic Core, and the Grant-in-Aid for Scientific Research by the Japanese Ministry of Education, Culture, Sports, Science and Technology.

The fourth and fifth authors were supported by Hanyang University, Korea, made in the program year of 2001.

References

[1] J.H. Conway and N.J.A. Sloane, Sphere Packing, Lattice and Groups (Third Edition). Springer-Verlag, New York, 1999.

[2] Z. Drezner and E. Erkut, Solving the continuous p-dispersion problem using non-linear programming. J. Oper. Res. Soc., **46** (1995), 516–520.

[3] M. Goldberg, Packing of 18 equal circles on a sphere. Elements of Mathematics, **20** (1965), 59–61.

[4] M. Goldberg, The packing of equal circles in a square. Mathematical Magazine, **43** (1970), 24–30.

[5] M. Goldberg, Packing of 14, 16, 17 and 20 circles in a circle. Mathematical Magazine, **44** (1971), 134–139.

[6] R.L. Graham, B.D. Lubachevsky, K.J. Nurmela and P.R.T. Östergård, Dense packing of congruent circles in a circle. Discrete Mathematics, **181** (1998), 139–154.

[7] D.-S. Kim, D. Kim and K. Sugihara, Voronoi diagram of a circle set from Voronoi diagram of a point set, I. Topology. Computer Aided Geometric Design, **18** (2001), 541–562.

[8] D.-S. Kim, D. Kim and K. Sugihara, Voronoi diagram of a circle set from Voronoi diagram of a point set, II. Geometry. Computer Aided Geometric Design, **18** (2001), 563–585.

[9] S. Kravitz, Packing cylinders into cylindrical containers. Mathematical Magazine, **40** (1967), 65–71.

[10] D.T. Lee and R.L. Drysdale, III, Generalization of Voronoi diagrams in the plane. SIAM J. Comput., **10** (1981), 73-87.

[11] B.D. Lubachevsky and R.L. Graham, Curved hexagonal packings of equal disks in a circle. Discrete and Computational Geometry, **18** (1997), 179–194.

[12] B.D. Lubachevsky and F.H. Stillinger, Geometric properties of random disk packing. J. Statist. Phys., **60** (1990), 561–583.

[13] M. Mollard and C. Payan, Some progress in the packing of equal circles in a square. Discrete Mathematics, **84** (1990), 303–307.

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

[15] A. Okabe, B. Boots, K. Sugihara and S.-N. Choi, Spatial Tessellations — Concepts and Applications of Voronoi Diagrams (2nd Edition). John Wiley and Sons, Chichester, 2001.

[16] G.E. Reis, Dense packing of equal circles within a circle. Mathematical Magazine, **48** (1975), 33–37.

[17] M. Sharir, Intersection and closest-pair problems for a set of planar disks. SIAM J. Comput., **14** (1985), 448–468.

[18] K. Sugihara and M. Iri, VORONOI2 Reference Manual—-Topology-oriented Version of the Incremental Methods for Constructing Voronoi Diagrams (2nd Edition). Dept. of Math. Eng., Univ. of Tokyo, 1993.

[19] K. Sugihara and M. Iri, A robust topology-oriented incremental algorithm for Voronoi diagrams. Internat. J. Computational Geometry and Applications, **4** (1994), 179–228.

[20] G. Valette, A better packing of ten circles in a square. Discrete Mathematics, **76** (1989), 57–59.

[21] C. Y. Yap, An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. Discrete and Computational Geometry, **2** (1987), 365–393.