The Problem of 4G Telecommunication Base Station Location

Deqing Fu, Wenyuan Ma, Wenxin Huang

High School Affiliated to Nanjing Normal University, Nanjing, China

Abstract: With the rapid development of 4G communication network, the position of base station is the key to the construction of the communication network. However, the distribution of base stations does not satisfy the growing demand, not only because of the low fraction of coverage, but also because of the inefficient overlap among the coverage areas of base stations, therefore contributing to dramatic waste and failing to increase the overall efficiency. This paper model the practical issue to the following mathematical problem: to cover a bounded area using unit circles, how many circles do we need at least? To solve this, on one hand, investigating from basic bounded graphs (e.g. circle, regular triangle and rectangle, etc.), calculates the upper bound of the number of unit circles used to cover a given basic bounded graph. On the other hand, this paper uses the greedy algorithm and the simulated annealing algorithm to discretize the map to a unit of points and then cover the unit of points, preferring the lower bound obtained by the simulated annealing algorithm, which is more optimal. At the end of the paper, we present a practical use. Taking Nanjing City as an example, we figure out a relatively optimal scheme to plant the base stations in Nanjing. Through the coverage of specific bounded graph, simulated annealing algorithm and the model of covering the whole plane, this paper concludes that if the coverage radius of each base station is 10 kilometers and the distance between pixels is 10, the optimal coverage number of unit circles has a range between 55 and 62. As far as Nanjing is concerned, it is clear that the latter two coverage schemes are better.

Key Words: coverage; base station; algorithm optimization.

Content

Chapter I Introduction

1.1 The Purpose of the Research

4G, as known as the 4th Generation of telecommunication, can satisfy the demand of nearly every costumers toward wireless services, have the ability to adeptly distribute resources, deal with constantly changing situations. 4G has a wide application due to its uncompetitive superiority.

However, consumers are not satisfied with the current condition of the poor signals of 4G telecommunications. In some public places, hardly can we capture any 4G signals, not to mention receiving and sending emails to work. Since the poor selections of base station locations, some potential consumers are not willing to start their 4G services whose development is far slower than the carriers and customers expected.

Undoubtedly, 4G telecommunication is the trend of global telecommunication technology while the selection of base stations need fund, labor and technology of the carriers. How to use fewer base stations to achieve the best results of user experience is the problem that needs to be solved.

An idea distribution of base stations needs following two requirements: No signal-blind area and fewer base stations to reduce the cost for the carriers.

Supposing that the covering area of a base station is a circular area with a certain radius. Thus, we can idealize the distribution plan to the following mathematical modeling: given a bounded area in a plane,S, using as fewer as possible bounder unit circles to cover S, satisfying $\forall u \in S, u$ is covered by a unit circle. How many unit circles are required at least? Where do these circles locate?

Using theoretical methods combined with algorithms of computer programming can solve this life problem.

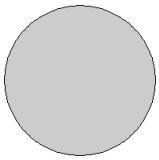
Using the mathematical modeling stated above, we will start with several bounded graph, hence conclude the upper and lower bound of requirement numbers of circles, and finally come out the optimized solution to cover Nanjing City.

1.2 Latest Progress and Research Status

If S is a bounded circular graph, using $R_n$to present radius of the biggest circular area n unit circles can cover. When $1 \leq n \leq 12$, former scientists such as Bezdek determine the value of R_n, as shown below.

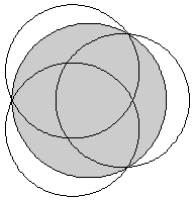
Circles Covering Circles[1] [2]

1.



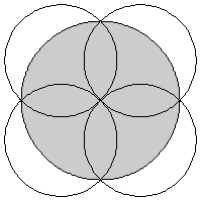
$R_1 = 1$

3.



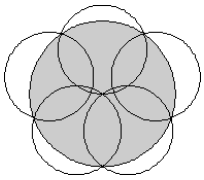
$R_3 = \frac{2\sqrt{3}}{3} = 1.154$

4.



$R_4 = \sqrt{2} = 1.414$

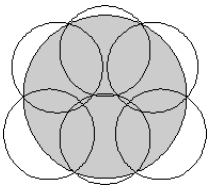
5.



$R_5 = 1.641$

Proved by Károly Bezdek in 1983

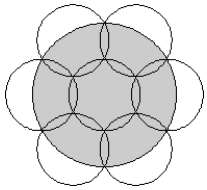
6.



$R_6 = 1.798$

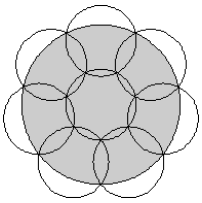
Proved by Károly Bezdek in 1979.

7.



$R_7 = 2$

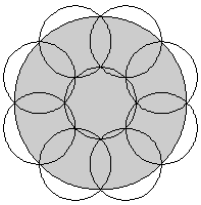
8.



$R_8 = 2.246$

Proved by Gábor Fejes Tóth in 1996.

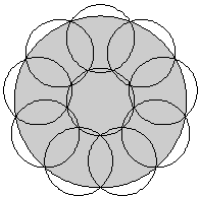
9.



$R_9 = \sqrt{2} + 1 = 2.414$

Proved by Gábor Fejes Tóth in 1996.

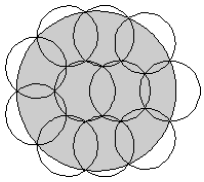
10.



$R_{10} = 2.532$

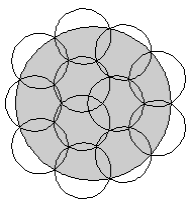
Found by D. Nagy in 1974

11.



$R_{11} = 2.631$

12.



$R_{12} = 2.769$

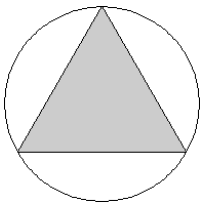
Found by Hans Melissen in 1997.

Found by Hans Melissen in 1997.

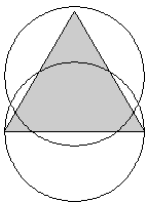
Figure 1-1

If S is a bounded triangular graph, using $S_n$to present the side length of the biggest triangular area n unit circles can cover. When $1 \leq n \leq 9$, former scientists such as Hans Melissen determine the value of S_n, as shown below.

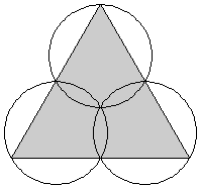
Circles Covering Triangles[3]

1. 

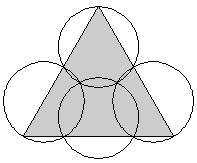
$s_1 = \sqrt{3} = 1.732$

2. 

$s_2 = 2$

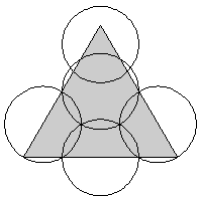
3. 

$s_3 = 2\sqrt{3} = 3.464$

4. 

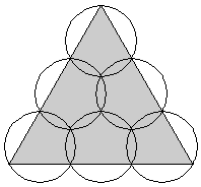
$s_4 = 2 + \sqrt{3} = 3.732$

Proved by Hans Melissen in 1997.

5. 

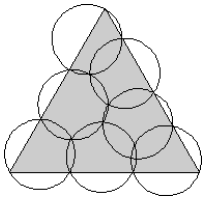
$s_5 = 4$

Proved by Hans Melissen in 1997.

6. 

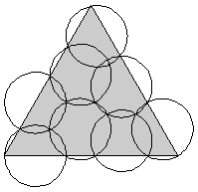
$s_6 = 3\sqrt{3} = 5.196$

Proved by Hans Melissen in 1997.

7. 

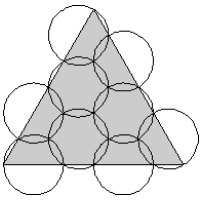
$s_7 = 5.398$

Found by P. Schuur in 1997.

8. 

$s_8 = 5.649$

Found by P. Schuur in 1997.

9. 

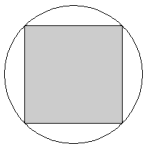
$s_9 = 6$

Proved by Hans Melissen in 1997.

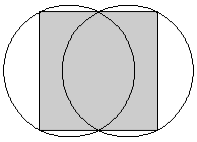
Figure 1-2

If S is a bounded square graph, using $q_n$to present the side length of the biggest square area n unit circles can cover. When $1 \leq n \leq 12$, former scientists such as Denes Nagy determine the value of q_n, as shown below.

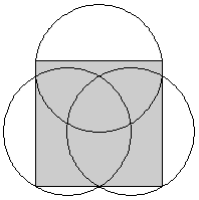
Circles Covering Squares [5]

1. 

$q_1 = \sqrt{2} = 1.414$

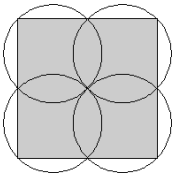
2. 

$q_2 = \frac{4\sqrt{5}}{5} = 1.788$

3. 

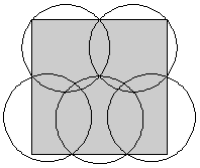
$q_3 = \frac{16\sqrt{65}}{65} = 1.984$

Proved by Denes Nagy in 1974.

4. 

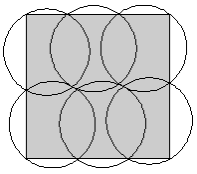
$q_4 = 2\sqrt{2} = 2.828$

Proved by Denes Nagy in 1974.

5. 

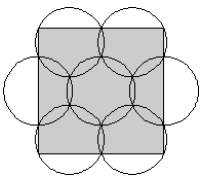
$q_5 = 3.065$

Proved by Hans Melissen [1] in 1997.

6. 

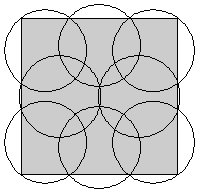
$q_6 = 3.347$

Found by Hans Melissen [1] in 1997.

7. 

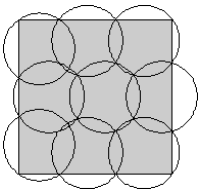
$q_7 = 1 + \sqrt{7} = 3.645$

Proved by Hans Melissen [1] in 1997.

8. 

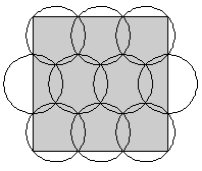
$q_8 = 3.841$

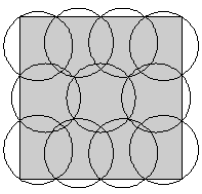
Found by Hans Melisse and P. Schuur [1] in 1997.

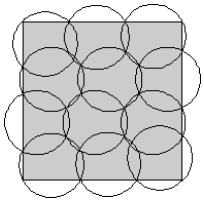
9. 

$q_9 = 4.335$

Found by Kari Nurmela and Patric Östergård in 2000.

10. 

11. 

12. 

$q_{10} = \frac{18+24\sqrt{3}}{13} = 4.582$	$q_{11} = 4.705$	$q_{12} = 4.943$
Found by T. Tarnai and Z Gáspár in 1995.	Found by Hans Melissen and P. Schuur[1] in 1997.	Found by Kari Nurmela and Patric Östergård in 2000.

Figure 1-3

Supposing the length of a rectangle is a, and width is 1and this rectangle can be covered by n congruent circles, we note that the least value of the radius of the required circles is $r_n(a)$, J. B. M. Melissen and P. C. Schuur give the following conclusions in reference [4]

Conclusion 1：If a≥$\frac{n}{\sqrt{3}}$, then $r_n(a)=\frac{1}{2n}\sqrt{a^2+n^2}$.

Conclusion 2：If n=2, then $r_2(a)=\frac{1}{4}\sqrt{a^2+4}$.

Conclusion 3：If n=3, then $r_3(a)=\begin{cases}\frac{\sqrt{16a^4+40a^2+9}}{16a}, & 1\le a<\frac{3}{2},\\ \frac{\sqrt{a^2+9}}{6}, & \frac{3}{2}\le a.\end{cases}$

Conclusion 4：If n=4, then $r_4(a)=\begin{cases}\frac{\sqrt{a^2+1}}{4}, & 1\le a<\sqrt{\frac{5+16\sqrt{10}}{15}},\\ \frac{2\sqrt{a^2+3}-a}{6}, & \sqrt{\frac{5+16\sqrt{10}}{15}}\le a<\frac{4\sqrt{3}}{3},\\ \frac{\sqrt{a^2+16}}{8}, & \frac{4\sqrt{3}}{3}\le a.\end{cases}$

Conclusion 5：If n=5, then

$$r_5(a)=\begin{cases}r_0, & 1\le a<2.0236230389,\\ \frac{\sqrt{5a^2-2a\sqrt{4a^2-9}}}{6}, & 2.0236230389\le a<\frac{\sqrt{510\sqrt{10}-375}}{15},\\ \frac{3\sqrt{a^2+8}-a}{16}, & \frac{\sqrt{510\sqrt{10}-375}}{15}\le a<\frac{5\sqrt{3}}{3},\\ \frac{\sqrt{a^2+25}}{10}, & \frac{5\sqrt{3}}{3}\le a,\end{cases}$$

While r_0 is the littlest solution of r in the following equation

$$65536r^6+8192(-4+5a^2)r^5+256(16-8a^2-7a^4)r^4-10240a^2r^3$$
$$+32a^2(64-52a^2-a^4)r^2-32a^4(16+5a^2)r+a^4(256+160a^2+94a^4)=0.$$

Conclusion 6：If n=6, then

$$r_6(a)=\begin{cases} \text{no solution} & , \quad 1\le a<\dfrac{2+\sqrt{5}}{2}, \\ \dfrac{4\sqrt{a^2+15}-a}{30} & , \quad \dfrac{2+\sqrt{5}}{2}\le a<2\sqrt{3}, \\ \dfrac{\sqrt{a^2+36}}{12} & , \quad 2\sqrt{3}\le a. \end{cases}$$

Conclusion 7：If n=7, then

$$r_7(a)=\begin{cases} \dfrac{\sqrt{4a^2+3}-a}{6} & , \quad 1\le a<\dfrac{\sqrt{30+24\sqrt{10}}}{3}, \\ \dfrac{3\sqrt{a^2+5}-2a}{10} & , \quad \dfrac{\sqrt{30+24\sqrt{10}}}{3}\le a<\dfrac{7\sqrt{3}}{3}, \\ \dfrac{\sqrt{a^2+49}}{14} & , \quad \dfrac{7\sqrt{3}}{3}\le a. \end{cases}$$

All the conclusions above have their own limits for they are too specific to give a general conclusion. When it comes to the coverage of bounded irregular region, there is no optimal way of coverage or optimization of general algorithm. Greedy algorithm is the only algorithm with discretization of the region, but it cannot achieve globally optimal value. In this paper, we analyze and study two ways to cover particular bounded figures with circles. Finally, we take Nanjing as an example, providing three methods to cover this irregular region (two ways of particular bounded figure coverage and a full-plane coverage) and compare the results of three methods. In the research of non-discretization model, we give out the circular iteration model and compare it with full-plane coverage model. Specifically, when covering rectangles with circles, it is easy to see our circular iteration model is an optimization of the Four-Leaf Clover Model. In the exploration of discretization method, we provide the improved greedy algorithm. Based on that, we further optimize the algorithm by simulated annealing algorithm and dichotomy. At last, we apply these two ways and full-plane coverage to Nanjing, obtain two relatively optimal results and compare them.

1.3 Prerequisite Knowledge: The Optimal Coverage of an Infinite Plane

Definition 1.31 n circles' covering efficiency in the full plane is $\varphi_n = \frac{S[\bigcup_{j=1}^{n} C_j]}{n \cdot \pi \cdot r^2}$, while C_j is the jth circle's covering scope and $S[\bigcup_{j=1}^{n} C_j]$ is the area of the covering scope of n circles, n is the total number of covering circles, r is the radius of a covering circle.

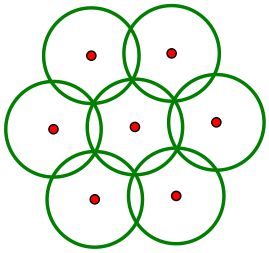


Figure 1-4

We find when covering the full plane, hexagonal-honeycomb coverage(Figure 1-4)can achieve the maximum covering efficiency.

Theorem 1.31 When covering the full plane, hexagonal-honeycomb coverage can achieve the maximum covering efficiency.

Proof First we consider two circles C_1, C_2 intersecting on the full-plane(like Figure 1-5). Regarding the point of intersection of C_1 and C_2, P, it is easy to see there should be at least another circle C_3 that covers P, so that the region surrounding P can be covered（like Figure 1-6）,or there is no uncovered region in the whole plane. Of course, more circles can be used to cover P. If n circles cover P, in order to maximize φ_n, n circles should intersect at P(take $n = 3$ as an example, like Figure 1-7)

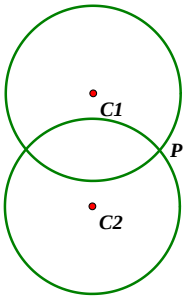


Figure 1-5

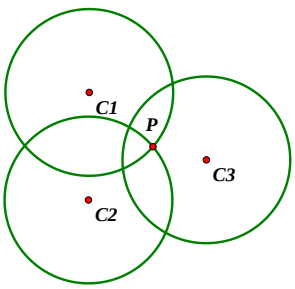


Figure 1-6

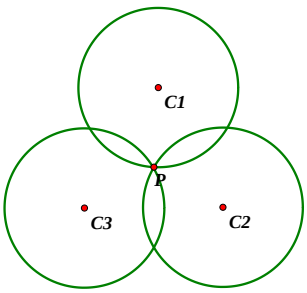


Figure 1-7

Theorem 1.32 if n circles intersect at P and the maximum of n circles' covering efficiency φ_n is $\max\{\varphi_n\}$, when n=3, $\max\{\varphi_n\}$ can achieve maximum 0.942331.

Proof Name covering circles $C_1, C_2, \ldots, C_n$ in clockwise, if the overlap area of C_i and C_{i+1}(C_n and C_1) is S_i(like Figure 1-8), covering efficiency

$$\varphi_n = \frac{n \cdot \pi \cdot r^2 - \sum_{i=1}^{n} S_i}{n \cdot \pi \cdot r^2}$$

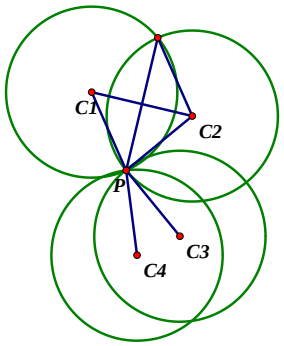
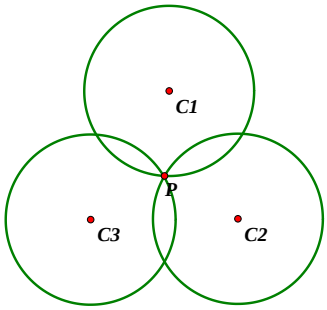


Figure 1-8



Figrue 1-9

Suppose $\angle C_i P C_{i+1} = \theta_i$,

then

$$S_i = r^2(\pi - \theta_i - \sin(\pi - \theta_i))$$

$$\sum_{i=1}^{n} S_i = r^2[(n-2)\pi - \sum_{i=1}^{n} \sin(\theta_i)]$$

In order that φ_n achieves maximum, $\sum_{i=1}^{n} \sin(\theta_i)$ should achieve maximum. According to Jensen ' s inequality, we should let θ_i all equal to $\frac{2\pi}{n}$. Under certain situation, $\max\{\varphi_n\} = \frac{2\pi + n \cdot \sin\frac{2\pi}{n}}{n \cdot \pi}$, this function decreases monotonically when $n \geq 3$. So when $n = 3$, $\max\{\varphi_n\}$ achieves its maximum, $\max\{\varphi_3\} = 0.942331$, like Figure 1-9.

With regard to any point of intersection of two covering circles, highest covering efficiency can be achieved by the way showed by Figure 1-9(which is the

part of hexagonal honeycomb). As a result, when it comes to full-plane coverage, hexagonal-honeycomb coverage can achieve the maximum covering efficiency.

So the optimal coverage of a full plane should be arranged like Figure 1-10.

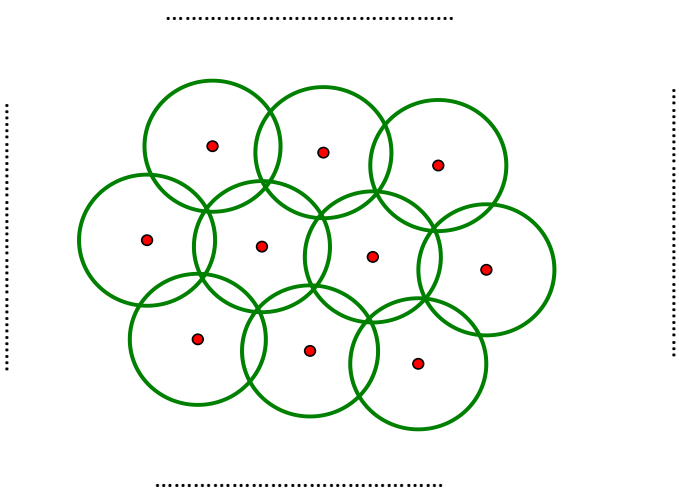


Figure 1-10

Chapter II The Coverage of Specific Bounded Graphs

2.1 The Coverage of Circle

Considering general conditions: for a circular area with a given radius, how many unit circles are needed to cover it thoroughly? We use two different iteration methods, and give out the programs to calculate the lower bound the number of needed circles as little as possible.

2.1.1 Iteration Inward

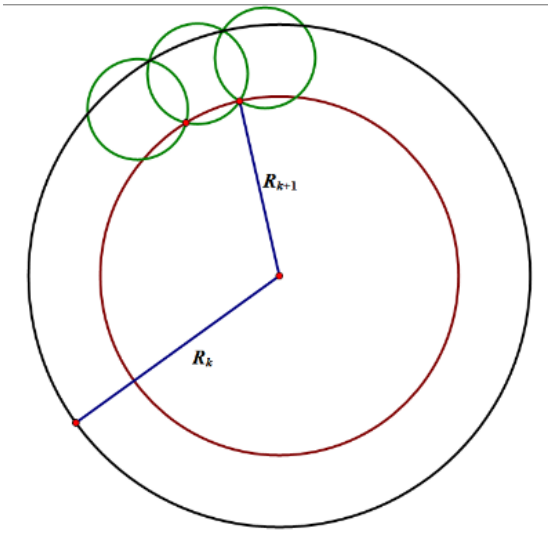


Figure 2-1

Supposing the radius of the given circular area is R. As shown in Figure 2-1, We take the method of iteration inward. To make the needed unit circles fewer, the part of the circumference of the given circular area each unit circle covers should be longer. The best way is to let the center of each unit circle lays on the circumference. Let us suppose when it comes the k-th iteration，The radius of the target circle is R_k,while $R_1 = R$.

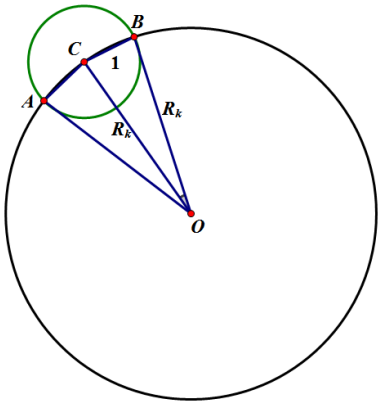


Figure 2-2

As shown in Figure 2-2,we can calculate that $\angle AOB = 2\cos^{-1}\left(1-\frac{1}{2R_k^2}\right)$。As the circumference of the bigger circle cannot always be the integral multiple of the length of $\widehat{AB}$, then the numbers of the needed unit circles in the k-th iteration is $C_k = \left\lceil \frac{\pi}{\cos^{-1}\left(1-\frac{1}{2R_k^2}\right)} \right\rceil$, while $\lceil x \rceil$ means rounding up，as known as the smallest integer not larger than x.

In order to guarantee that each unit circle can cover the longer length of the circumference of the bigger circle, we come up with another problem that the circumference of the bigger circle is not always the integral multiple of the longest length each unit circle can cover, then at least one circle may be overlapped or overlapping other circles of a great waste, thus the circle of the next iteration will not be the optimal one, meaning that the radius R_{k+1}may be larger than it should be if optimized. Then we do the following adaptation: let each unit circle cover the equal length of the bigger circle, then each cover the angle of $\angle AOB = \frac{2\pi}{C_k}$. Let $\beta_k = \angle AOC = \frac{1}{2}\angle AOB = \frac{\pi}{C_k} = \frac{\pi}{\left\lceil \frac{\pi}{\cos^{-1}\left(1-\frac{1}{2R_k^2}\right)} \right\rceil}$.

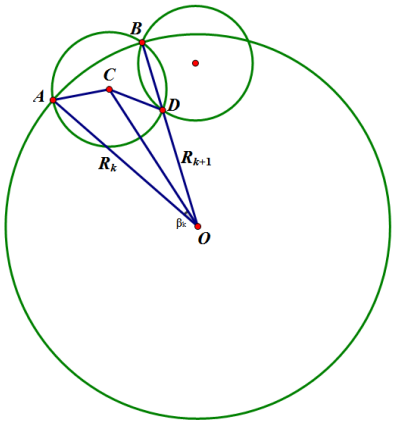


Figure 2-3

Respectively using cosine theorem to ΔAOC and ΔCOD, we can get the following iteration equation:

$$R_{k+1} = 2R_k\cos^2(\beta_k) - 2\cos\beta_k\sqrt{1 - R_k^2\sin^2(\beta_k)} - R_k$$

While $\beta_k = \frac{\pi}{\left\lceil \frac{\pi}{\cos^{-1}\left(1-\frac{1}{2R_k^2}\right)} \right\rceil}$。

When it comes an R_{n+1} satisfying $0 < R_{n+1} < 1$, then the number of needed circles is

$$C_{in} = 1 + \sum_{k=1}^{n} C_k = 1 + \sum_{k=1}^{n} \left\lceil \frac{\pi}{\cos^{-1}\left(1 - \frac{1}{2R_k^2}\right)} \right\rceil$$

When it comes an $R_{n+1} = 0$, then the number of needed circles is

$$C_{in} = \sum_{k=1}^{n} C_k = \sum_{k=1}^{n} \left\lceil \frac{\pi}{\cos^{-1}\left(1 - \frac{1}{2R_k^2}\right)} \right\rceil$$

C_{in} represents the total number of needed unit circles using Iteration Inward Programs shown in Appendix 1.

Using the algorithm shown above and Typing in the maximum number of the radius using n（1~12）circles listed in Chapter I，Output the needed number of circles and compare with original data in Chapter I. We get the table below.

Maximum Radius	Basic number in Chapter I	Number of circles by Program	Deviation	Deviation Rate
1	1	1	0	0
1.154	3	4	1	0.333333
1.414	4	5	1	0.25
1.641	5	6	1	0.2
1.798	6	7	1	0.166667
2	7	8	1	0.142857
2.246	8	8	0	0
2.414	9	9	0	0
2.532	10	13	3	0.3
2.631	11	13	2	0.181818
2.769	12	14	2	0.166667

Table 2-1

（Deviation Rate= Deviation/Basic number in Chapter I）

Number of Circles by program means the result generated by the method discussed above.

Using Table 2-1, we graph the chart of deviation rate above.

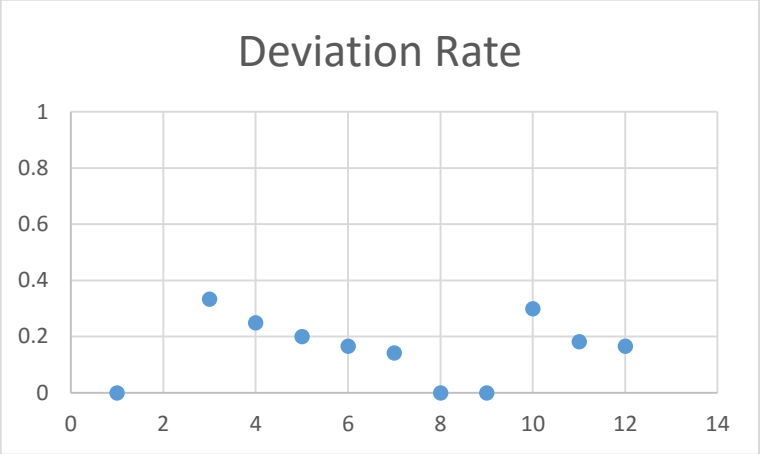


Chart 2-1

From the chart above we can see that the deviation rate has a pattern of period, and the maximum number of each period has a trend of decreasing. This means that when the radius of the given circular area is bigger enough, the deviation rate can be too small to be counted. So the methods is valid.

2.1.2 Iteration Outward

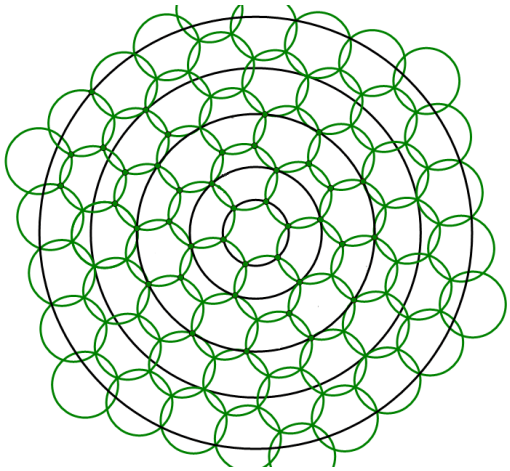


Figure 2-4

As shown in Figure 2-4, we start to consider the method of iteration outward to cover a given circular area.

Firstly, we consider some specific radius of the target circular area.

As is known to all, the largest radius of circle that t rounds of unit circles can cover is:

When t is an even, $t = 2k$.Then

$$R_t = 2 + 3(k-1) = 3k - 1$$

When $t$is an odd, $t = 2k - 1$.Then

$$R_t = R_{2k-1} = \sqrt{\frac{3}{4} + \left(\frac{1}{2} + 3(k-1)\right)^2}$$

While R_t represents the largest radius of circluar area t rounds of unit circles can cover.

When iterating outward, as there is one more round, there comes $6t$ unit circles.

Thus we have the iteration equation below [7]:

$$C_{t+1} = C_t + 6t$$

$$C_1 = 1$$

Solve the equation, we get that the number of unit circles on the t-th round is

$$C_t = 3t^2 - 3t + 1$$

Secondly, if the radius of the target circle, R, satisfying $R_{n-1} < R \leq R_n$, then we need

$$C_{out} = \sum_{t=1}^{n} C_t = \sum_{t=1}^{n} (3t^2 - 3t + 1) = n^3 + n^2 + n$$

circles.

Program is shown in Appendix 2

Then, we compare the two methods of iteration

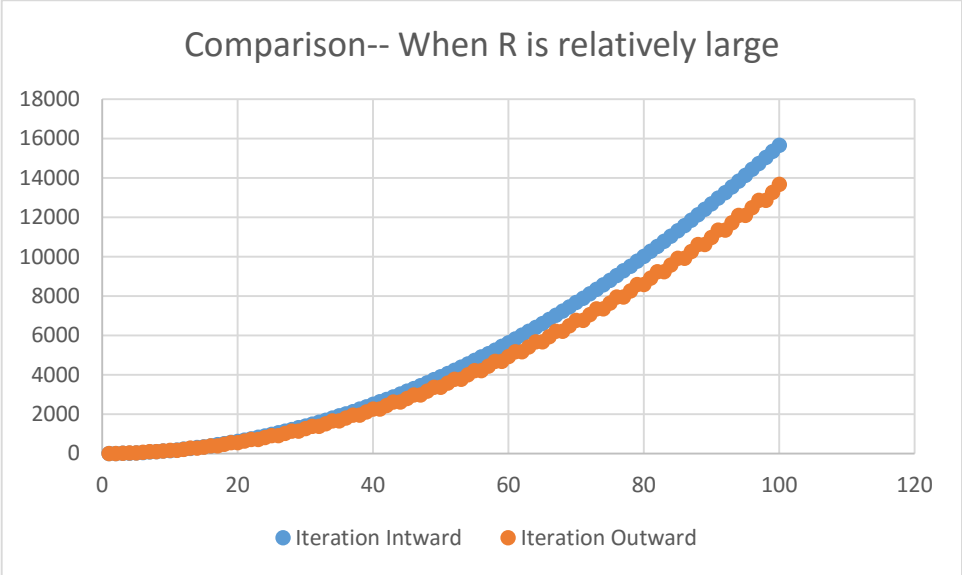


Chart 2-2

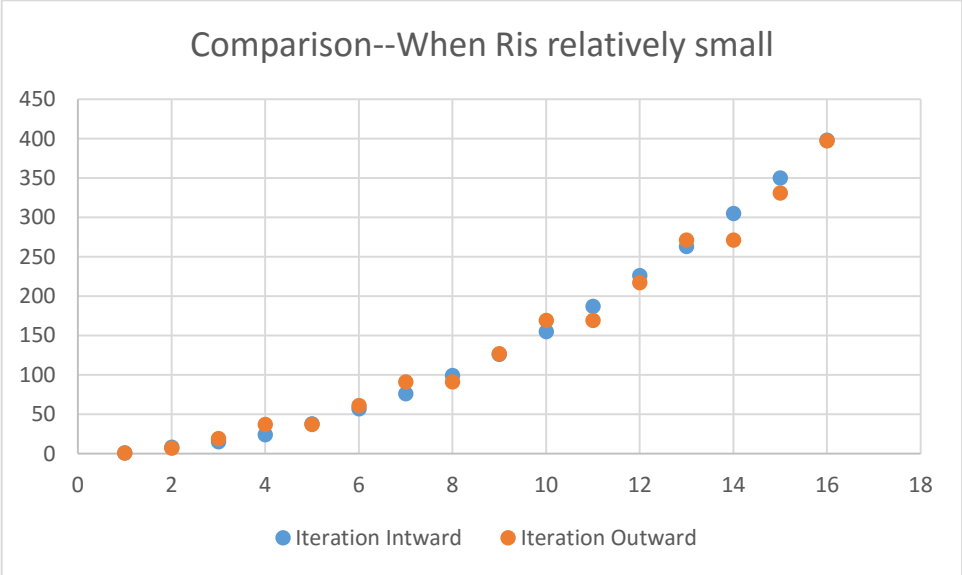


Chart 2-3

Comparing these two methods we find out that when R is relatively small, each one has its own advantage but when R is relatively large, the method of Iteration Outward is far better.

Theorem 2.1: The number of unit circles to cover a given circular area is

$$C \leq \min\{C_{in}, C_{out}\}$$

2.2 The Coverage of Triangle

We only consider the problem of equilateral triangles using the same progression as that of the problem of the coverage of circles.

2.2.1 Circular Iteration

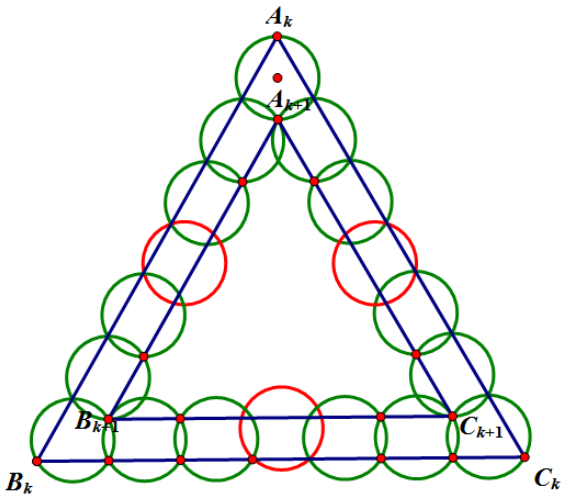


Figure 2-5

As shown in Figure 2-5, supposing the side length of the given equilateral triangle is a. Using similar reasoning as we do for the coverage of circle, we do what we can to make the side length of $\Delta A_{k+1}B_{k+1}B_{k+1}$ as small as possible. Then we get the iteration equation below:

Excluding the three circles on the acme of the triangle, the number of circles on each side is:

$$T_{c_k} = \left\lceil \frac{a_k - 2\sqrt{3}}{\sqrt{3}} \right\rceil$$

While a_k means the side length of the triangle in the k − th iteration,

$$a_1 = a$$

And

$$a_{k+1} = a_k - 4\sqrt{3}\sin(\alpha_k)$$

In which,

$$\alpha_k = \cos^{-1}\left(\frac{a_k - 2\sqrt{3}}{2\left\lceil\frac{a_k-2\sqrt{3}}{\sqrt{3}}\right\rceil}\right)$$

When $0 \le a_{n+1} \le \sqrt{3}$,

$$T_{c_0} = \sum_{k=1}^{n}(3T_{c_k} + 3)$$

When $a_{n+1} = 0$,

$$T_c = T_{c_0} = \sum_{k=1}^{n}(3T_{c_k} + 3)$$

When $a_{n+1} \neq 0$,

$$T_c = T_{c_0} + 1 = 1 + \sum_{k=1}^{n}(3T_{c_k} + 3)$$

In which , $T_c$means the total number of unit circles in the Circular Iteration.

Program is shown in Appendix 3.

Analysis of deviation: similarly, we can get the Table 2-2.

Maximum Length	Basic number of circles	Circles by Program	Deviation	Deviation Rate
1.732	1	1	0	0
2	2	3	1	0.5
3.464	3	3	0	0
3.732	4	6	2	0.5
4	5	6	1	0.2
5.196	6	7	1	0.166667
5.398	7	9	2	0.285714
5.649	8	9	1	0.125
6	9	10	1	0.111111

Table 2-2

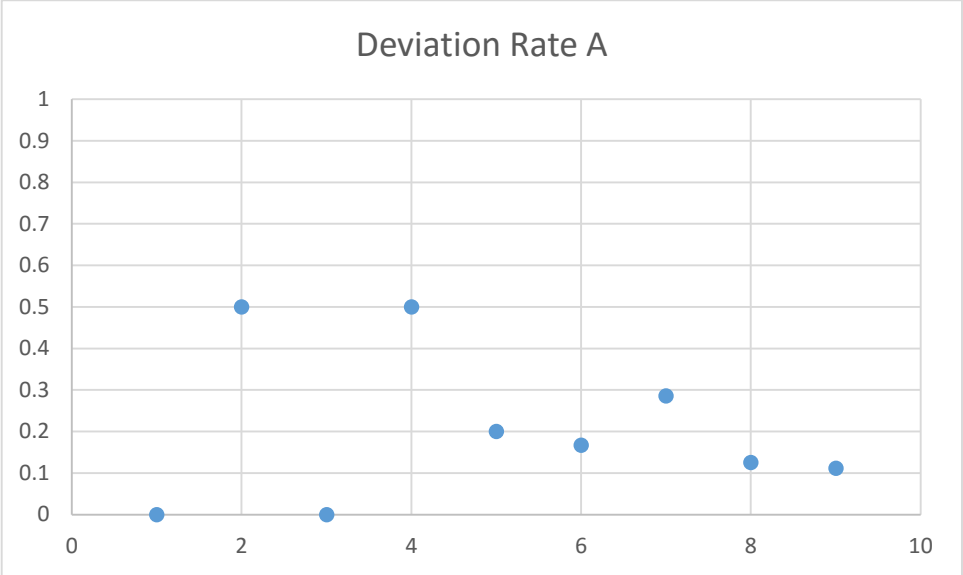


Chart 2-4

2.2.2 Linear Iteration

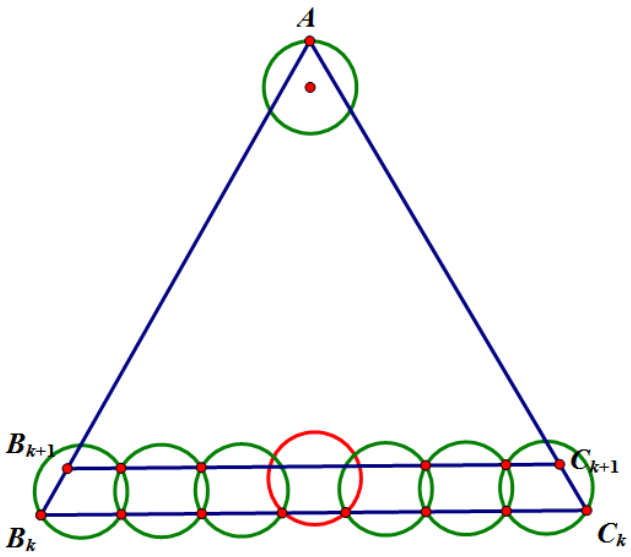


Figure 2-6

As shown in Figure 2-6, supposing the side length of the given equilateral triangle is a. Considering one side of the target triangle, we are going to optimize in order to make the length of $B_{k+1}C_{k+1}$ as small as possible.

The number of circles on each row is

$$T_{l_k} = \left\lceil \frac{a_k - 2\sqrt{3}}{\sqrt{3}} \right\rceil + 2$$

And $a_1 = a$

$$a_{k+1} = a_k - \frac{4}{\sqrt{3}}\sin(\alpha_k)$$

In which,

$$\alpha_k = \cos^{-1}\left(\frac{a_k - 2\sqrt{3}}{2\left\lceil\frac{a_k - 2\sqrt{3}}{\sqrt{3}}\right\rceil}\right)$$

When $0 \le a_{n+1} < 2\sqrt{3}$

$$T_{l_0} = \sum_{k=1}^{n} T_{l_k}$$

If $a_{n+1} = 0$,

$$T_l = T_{l_0} = \sum_{k=1}^{n} T_{l_k}$$

If $0 < a_{n+1} \le \sqrt{3}$

$$T_l = 1 + T_{l_0} = 1 + \sum_{k=1}^{n} T_{l_k}$$

If $\sqrt{3} < a_{n+1} \le 2$

$$T_l = 2 + T_{l_0} = 2 + \sum_{k=1}^{n} T_{l_k}$$

If $2 < a_{n+1} \le 2\sqrt{3}$

$$T_l = 3 + T_{l_0} = 3 + \sum_{k=1}^{n} T_{l_k}$$

In which, T_l means the total number of circles needed to cover the target triangle.

Program is shown in Appendix 4.

Analysis of deviation

Maximum Side Length	Number of basic circles	Circles by Program	Deviation	Deviation Rate
1.732	1	1	0	0
2	2	2	0	0
3.464	3	3	0	0
3.732	4	4	0	0
4	5	5	0	0
5.196	6	8	2	0.333333
5.398	7	7	0	0
5.649	8	8	0	0
6	9	10	1	0.111111

Table 2-3

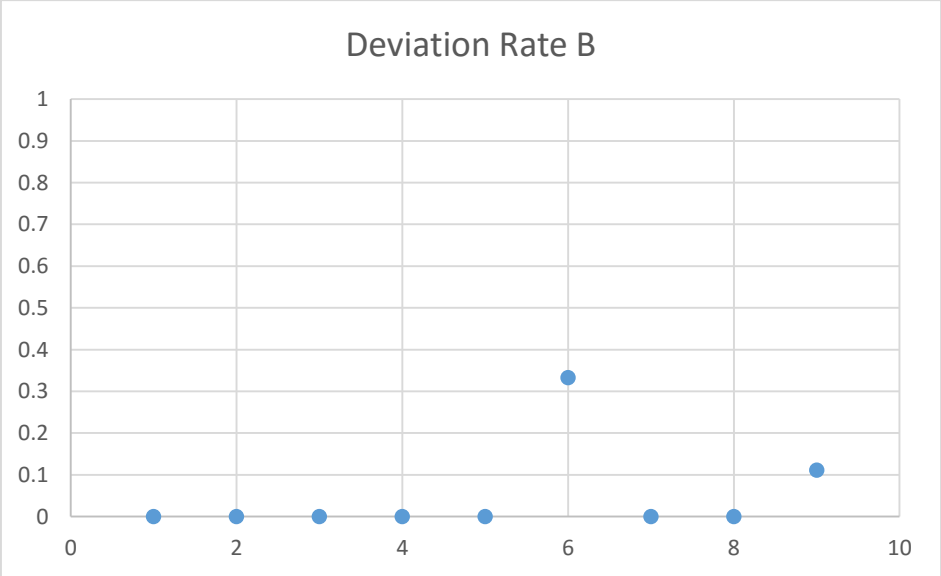


Chart 2-5

From the chart above, we can see that Linear Iteration has a smaller deviation rate. Meanwhile, we use the same means of comparing coverage of circle to compare the deviation of the two methods of covering triangle.

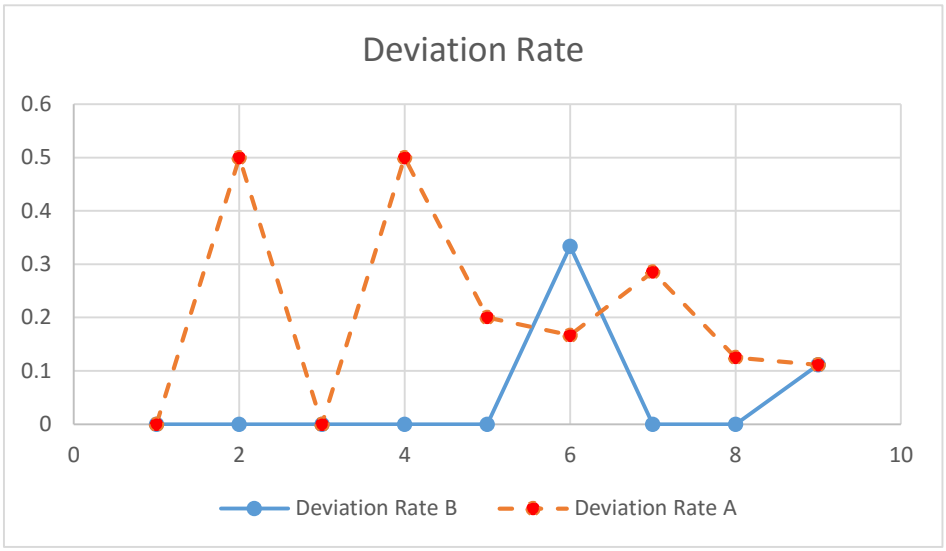


Chart 2-6

Then we use the programs in Appendix 3 and 4, inputting the integers from 1 to 100 as the side length of the target triangle, then we calculate the difference between two methods, and get the chart below:

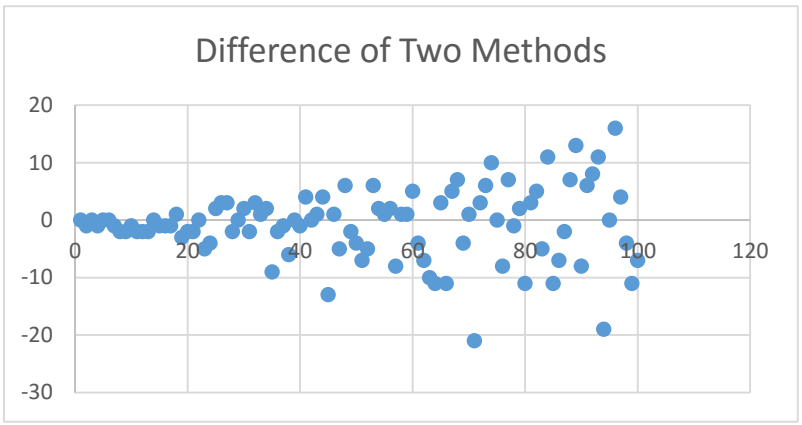


Chart 2-7

We can find out that the dots equally distribute above and below the x-axis, meaning that each method has its own advantage due to the side length of the target triangle.

Theorem 2.2: The number of unit circles to cover a given triangular area is

$$T \leq \min\{T_c, T_l\}$$

2.3 The Coverage of Rectangle

We use the same progression as that of the problem of the coverage of circles.

2.3.1 Circular Iteration, Method 1

In this method, we are going to make the newly generated rectangle after one iteration as smaller as possible. Supposing the length and width of the target rectangular area are a,b, we optimize the Four-Leaf Clover Model [4] (shown in Figure 2-7) and get the iteration relation equation below.

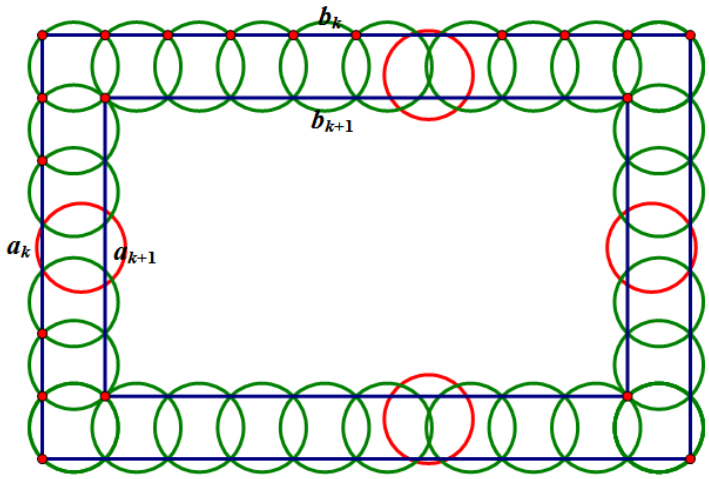


Figure 2-7

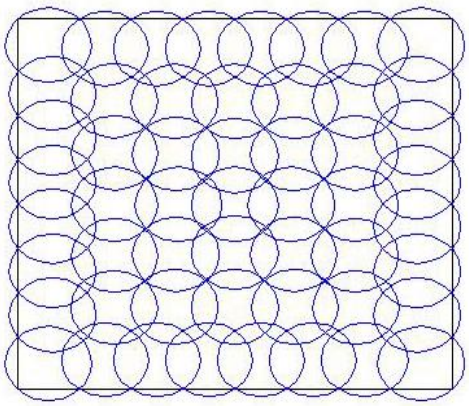


Figure 2-8

As shown in Figure 2-7, the four circles that are marked with red are the ones that need to be optimized. We use the same method as stated in the Coverage of Circle section and get the iteration equation below:

$$a_{k+1} = a_k - 2 \cdot \frac{(\cos(\beta_k) + \sin(\beta_k))(\cos(\beta_k) + \frac{\sqrt{2}}{2})}{\sqrt{2} + \cos(\beta_k) - \sin(\beta_k)}$$

$$b_{k+1} = b_k - 2 \cdot \frac{(\cos(\alpha_k) + \sin(\alpha_k))(\cos(\alpha_k) + \frac{\sqrt{2}}{2})}{\sqrt{2} + \cos(\alpha_k) - \sin(\alpha_k)}$$

In which, a_{k+1}means the length of the newly generated rectangle at the k $-$ th iteration,

$a_1 = a.$

b_{k+1}means the width of the newly generated rectangle at the $k-$ th iteration,

$b_1 = b.$

And

$$\alpha_k = \cos^{-1} \frac{a_k - 2\sqrt{2}}{2\left\lceil \frac{a_k}{\sqrt{2}} - 2 \right\rceil}$$

$$\beta_k = \cos^{-1} \frac{b_k - 2\sqrt{2}}{2\left\lceil \frac{b_k}{\sqrt{2}} - 2 \right\rceil}$$

In which, α_k, β_k are shown in Figure 2-9

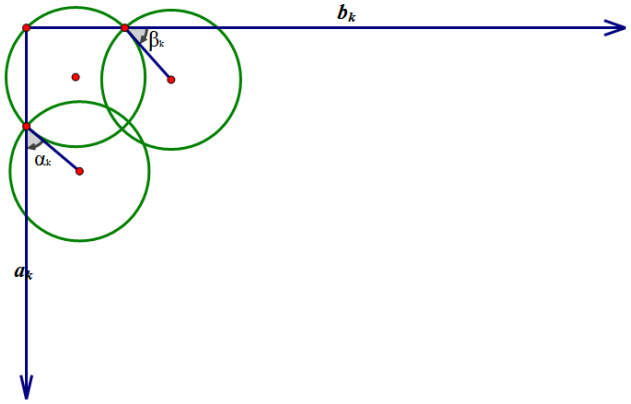


Figure 2-9

The total number of circles used to cover each round is

$$R_{in_k} = \left(\left\lceil \frac{a_k}{\sqrt{2}} \right\rceil + \left\lceil \frac{b_k}{\sqrt{2}} \right\rceil - 4\right) \cdot 2 + 4 = 2\left\lceil \frac{a_k}{\sqrt{2}} \right\rceil + 2\left\lceil \frac{b_k}{\sqrt{2}} \right\rceil - 4$$

When it comes to an situation where either a_{n+1} or $b_{n+1}$is smaller than $2\sqrt{2}$, we have

$$R_{in_0} = \sum_{k=1}^{n} R_{in_k}$$

Then the total number of needed circles is

$$R_{in} = R_{in_0} + \left\lceil \frac{a_n}{\sqrt{2}} \right\rceil \left\lceil \frac{b_n}{\sqrt{2}} \right\rceil = \sum_{k=1}^{n} R_{in_k} + \left\lceil \frac{a_n}{\sqrt{2}} \right\rceil \left\lceil \frac{b_n}{\sqrt{2}} \right\rceil$$

Program is shown in Appendix 5.

Deviation Analysis: As the former scientist only dealt with problems of squares, so we let a=b to do the deviation analysis.

Maximum Side Length	Number of Basic Circles	Number of Circles by Program	Deviation	Deviation Rate
1.414	1	1	0	0
1.788	2	4	2	1
1.984	3	4	1	0.333333
2.828	4	4	0	0
3.065	5	8	3	0.6
3.347	6	8	2	0.333333
3.645	7	8	1	0.142857
3.841	8	9	1	0.125
4.335	9	13	4	0.444444
4.582	10	13	3	0.3
4.705	11	13	2	0.181818
4.943	12	16	4	0.333333

Table 2-4

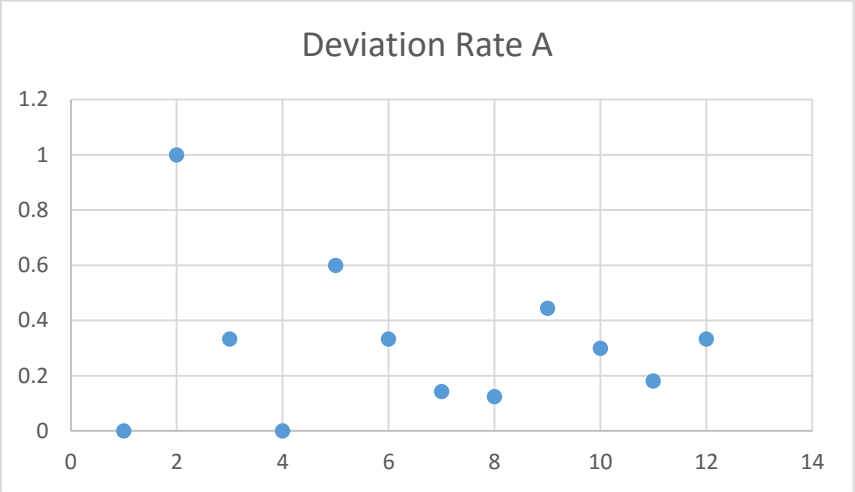


Chart 2-8

2.3.2 Circular Iteration, Method 2

In this method, we are going to let the circles on the circumference of the target rectangle in the k-th iteration, as fewer as possible.

Similarly to the method of optimizing the coverage of triangle and circle, we are going to optimize the circles marked red in Figure 2-10.

Then we get:

$$A_{out_k} = \left\lceil \frac{a_k - 2\sqrt{2}}{2} \right\rceil$$

$$B_{out_k} = \left\lceil \frac{b_k - 2\sqrt{2}}{2} \right\rceil$$

In which, at the k-th iteration

A_{out_k} represents the number of circles on the length.

B_{out_k} represents the number of circles on the width.

And

$$a_{k+1} = a_k - 2 \cdot \frac{(\cos(\beta_k) + \sin(\beta_k))(\cos(\beta_k) + \frac{\sqrt{2}}{2})}{\sqrt{2} + \cos(\beta_k) - \sin(\beta_k)}$$

$$b_{k+1} = b_k - 2 \cdot \frac{(\cos(\alpha_k) + \sin(\alpha_k))(\cos(\alpha_k) + \frac{\sqrt{2}}{2})}{\sqrt{2} + \cos(\alpha_k) - \sin(\alpha_k)}$$

In which, at the k-th iteration

a_{k+1} and b_{k+1} represent respectively the length and width of the newly generated rectangle. And

$$a_1 = a$$

$$b_1 = b$$

$$\alpha_k = \cos^{-1}\frac{a_k - 2\sqrt{2}}{2\left\lceil\frac{a_k-2\sqrt{2}}{2}\right\rceil}$$

$$\beta_k = \cos^{-1}\frac{b_k - 2\sqrt{2}}{2\left\lceil\frac{b_k-2\sqrt{2}}{2}\right\rceil}$$

In which, α_k, $\beta_k$are shown in Figure 2-11

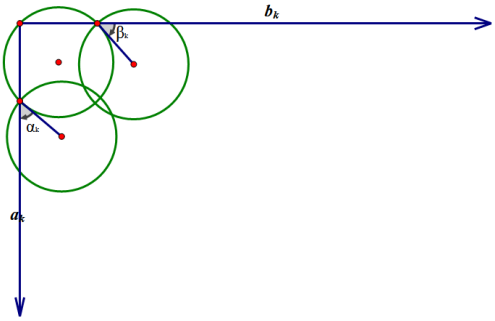


Figure 2-11

The number of circles of k-th round of the k-th iteration is

$$R_{out_k} = \left(\left\lceil\frac{a_k - 2\sqrt{2}}{2}\right\rceil + \left\lceil\frac{b_k - 2\sqrt{2}}{2}\right\rceil\right)\cdot 2 + 4$$

When it iterates to an end in which either a_{n+1} or b_{n+1} is smaller than $2\sqrt{2}$,

$$R_{out_0} = \sum_{k=1}^{n} R_{out_k}$$

The total number is

$$R_{out} = N_0 + \left\lceil\frac{a_n}{\sqrt{2}}\right\rceil\left\lceil\frac{b_n}{\sqrt{2}}\right\rceil = \sum_{k=1}^{n} R_{out_k} + \left\lceil\frac{a_n}{\sqrt{2}}\right\rceil\left\lceil\frac{b_n}{\sqrt{2}}\right\rceil$$

Program is shown in Appendix 6.

Deviation Analysis:

Maximum Length	Number of Basic Circles	Number of circles By Program	Deviation	Deviation Rate
1.414	1	1	0	0
1.788	2	4	2	1
1.984	3	4	1	0.333333
2.828	4	4	0	0
3.065	5	8	3	0.6
3.347	6	9	3	0.5
3.645	7	9	2	0.285714
3.841	8	9	1	0.125
4.335	9	12	3	0.333333
4.582	10	12	2	0.2
4.705	11	12	1	0.090909
4.943	12	16	4	0.333333

Table 2-5

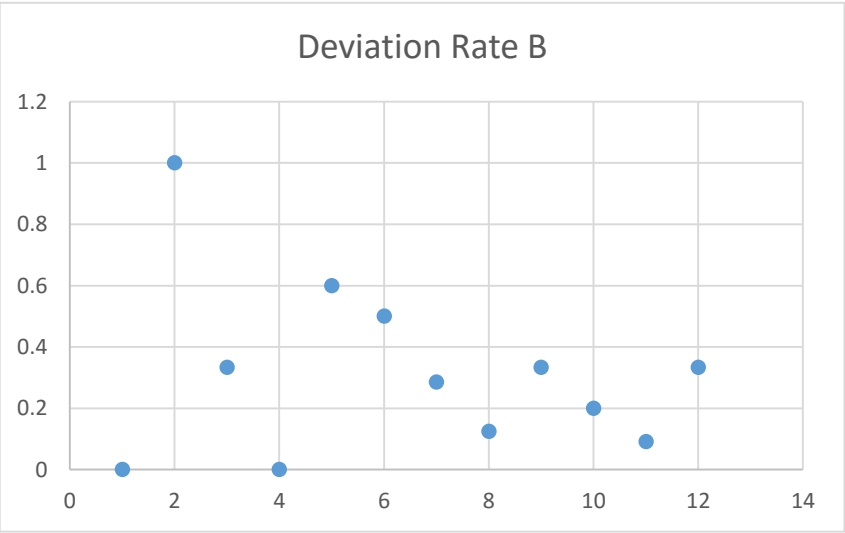


Figure 2-9

Then we compare the two methods of Circular Iteration and calculate the difference.

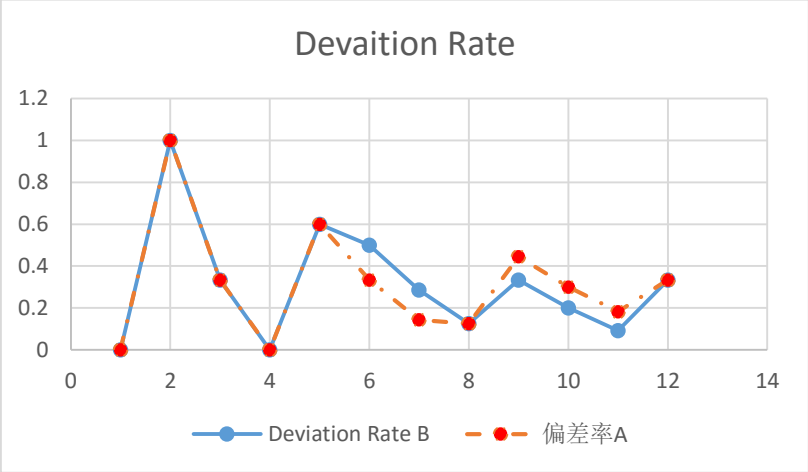


Chart 2-10

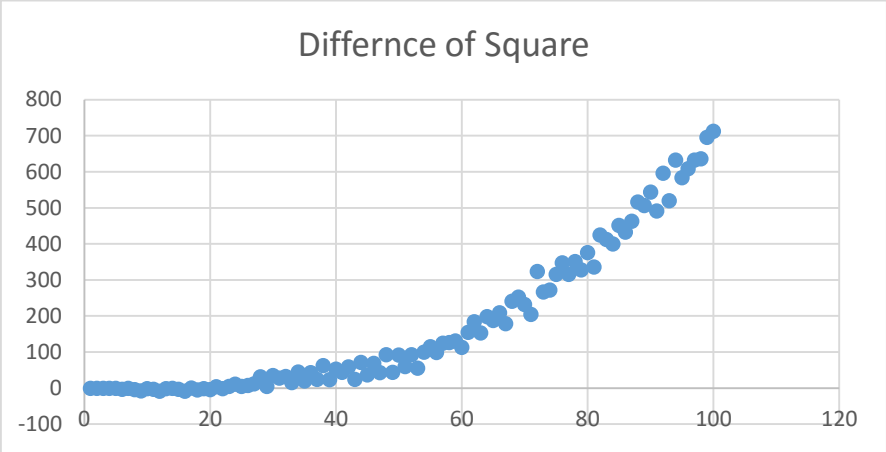


Chart 2-11

From the charts above, we find out that Method 1 is way better.

So let's mark $\boldsymbol{R_c = R_{out}}$.

2.3.3 Linear Iteration

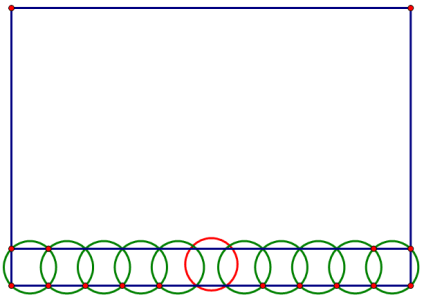


Figure 2-12

By optimizing the circle marked red in the Figure 2-12, we can easily calculate the needed number of circles of Linear Iteration.

$$R_l = \left\lceil \frac{a}{\frac{(\cos\beta+\sin\beta)(\cos\beta+\frac{\sqrt{2}}{2})}{\sqrt{2}+\cos\beta-\sin\beta}} \right\rceil \cdot \left\lceil \frac{b}{\sqrt{2}} \right\rceil$$

In which,

$$\beta = \cos^{-1}\frac{b-2\sqrt{2}}{2\left\lceil \frac{b}{\sqrt{2}}-2 \right\rceil}$$

And β has its geographic meaning shown in Figure 2-13

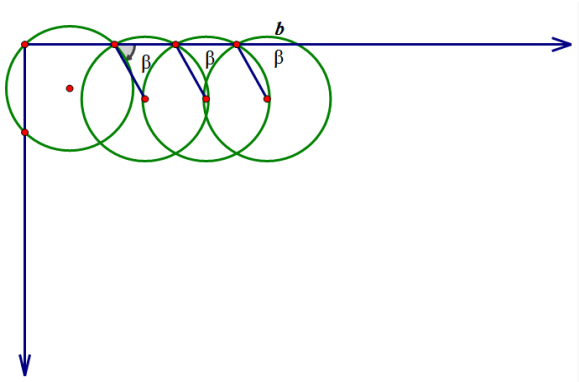


Figure 2-13

Program shown in Appendix 7.

Similarly, we calculate the difference between linear iteration and circular iteration.

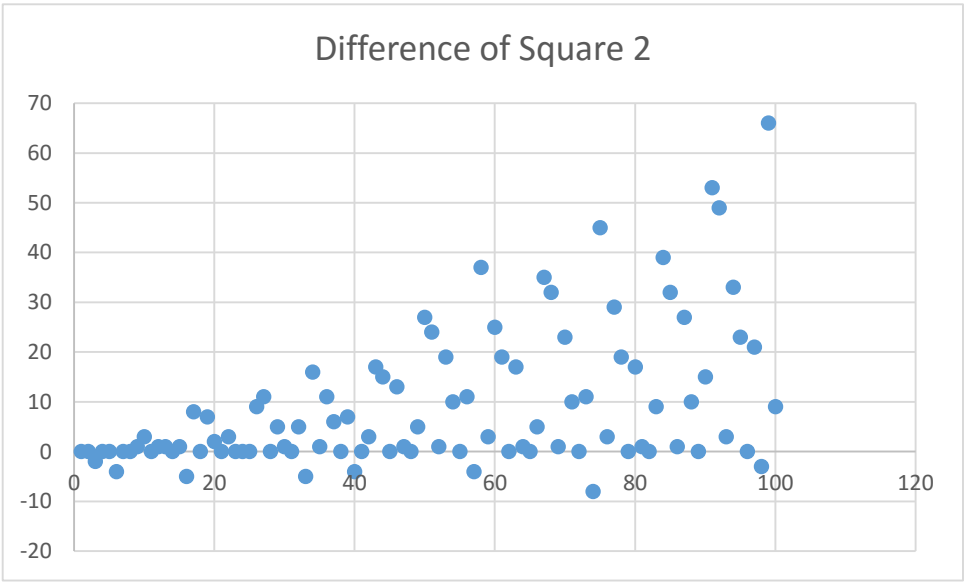


Chart 2-12

Theorem 2.3: The number of unit circles to cover a given rectangular area is

$$\boldsymbol{R \le \min\{R_c, R_l\}}$$

Chapter III Algorithm Optimization

3.1 Estimate the Lower Bound with Greedy Algorithm

algorithm analysis and realization：

nomenclature： P the set of uncovered points

P_i the ith point

N the number of points

C the set of covering circles' centers

C_j the center of the jth circle

K the number of covering circles

S_j the set of all the points covered by the circle whose center is C_j

R radius of a covering circle

$dist(P_i, C_j)$ the distance between point P_i and point C_j

$delete\_rate$ the ratio between the number of circles deleted every time and the total number of circles in current solution

Modeling：

In order to solve the covering problem mentioned above more conveniently, we can discretize the uncovered region into several points $P_i(1 \le i \le N)$，and simplify the problem to obtain a covering method C(so that $\forall P_i, \exists C_j, dist(P_i, C_j) \le R$,while $1 \le j \le K, C_j \in P$). If we put all the points that can be covered by C_j into a set S_j, it is easy to see it is a set coverage problem. Unfortunately, set coverage problem is NP-hard problem, which means it does not have a polynomial-time algorithm unless $P = NP$. To approximate the solution of this problem, academics have proposed a greedy algorithm as follows,

step 1： $V = \emptyset, C = \emptyset, K = 0$.

step 2： $V = \underset{1 \le j \le N}{max}\{V \cup S_j\}$, $C = C \cup P_j, K = K + 1$.

step 3： if $V = P$, turn to step 4， else turn to step 2.

step 4：print solution set C.

Although this algorithm is effective in solving set coverage problem, it may "be trapped" in regional optimal value when dealing with graph covering problem, leading to much waste. As a result, we introduced a mechanism of random "backward" and improved the traditional algorithm.

step 1：$V = \emptyset, C = \emptyset, delete\_rate = 0.7, tempK = 0, Temp = \emptyset, t = 1000000.$

step 2：$V = \underset{1 \le j \le N}{max}\{V \cup S_j\}$，$tempK = tempK + 1, Temp = Temp \cup P_j$.

step 3：if $V = P$, then turn to step 4, else turn to step 2.

step 4：if $tempK < K$, then $K = tempK, C = Temp.$

step 5: delete $K \cdot delete\_rate$ elements from both V and $Temp$, $K = K - K \cdot delete\_rate.$

step 6：repeat step 2 to step 5 for t times.

step 7：print solution set C.

In the improved greedy algorithm, every time we get a feasible solution, we delete some circles from this solution and do greedy choosing again so that we can avoid choosing the optimal state in every decision and missing the globally optimal solution.

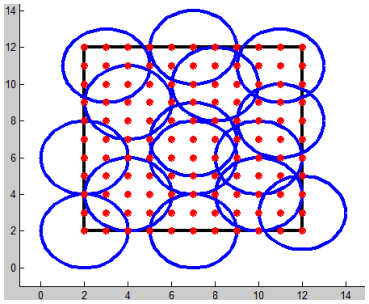
Theorem 3.1 In the algorithm above, every time after deleting some circles from $Temp$, $\exists C_j$, while $1 \le j \le K$, is not the regional optimal solution.

Proof Suppose $C_1, C_2, C_3 \cdots C_K$ are points chosen in a greedy algorithm in chronological order, we can know in the ith decision, C_i is more optimal than $C_{i+1}, C_{i+2} \ldots C_K$, so after deleting $[K \cdot delete\_rate]$ circles, there is a possibility of $(1 - \frac{1}{\binom{k}{[k \cdot delete]}})$, $\exists C_j$, so that $j > i, C_i$ is the center of a deleted circle. In the ith decision, C_i is a choice more optimal than C_j.

The MATLAB program is in Appendix 8

The following graphs compare the traditional greedy algorithm and improved greedy algorithm. Take circles with radius of 2 covering a 10×10 square as an example. We discretize the square into regular 1×1 net points(The red points in the graph)

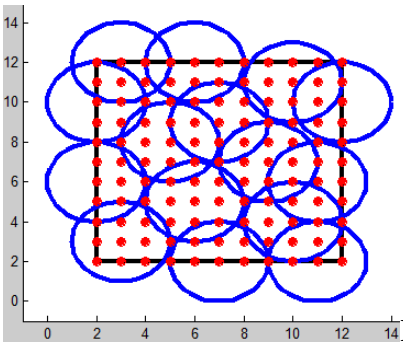
traditional greedy algorithm:



K=15

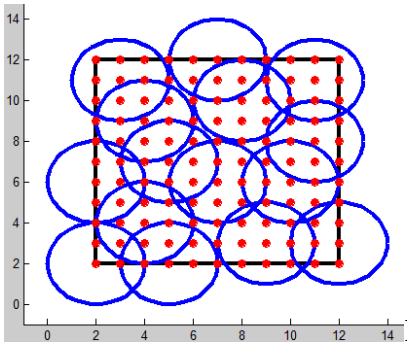
Figure 3-1

try improved greedy algorithm for several times:



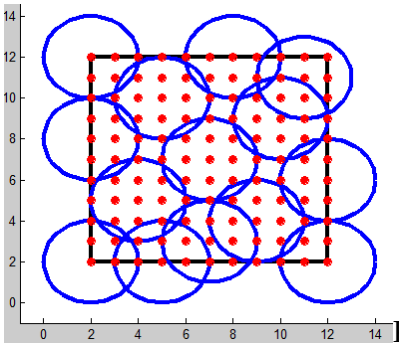
K=15

Figure 3-2



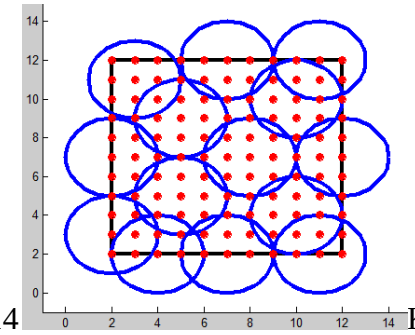
K=15

Figure 3-3



K=14

Figure 3-4



K=14

Figure 3-5

3.2 Optimize with Simulated Annealing Algorithm

Although improved greedy algorithm can "jump out of " regional optimal value to a certain extent, it is too arbitrary and lacks robustness. What's more, the model above simplifies uncovered points and circle centers into one set of points, leading to much waste and incongruence to reality. So we use simulated annealing algorithm, which is more effective, to solve this problem.

Simulated Annealing algorithm, based on Monte-Carlo iterative strategy, is a random optimizing algorithm. It originates in the similarity between annealing process in physics and optimization problem. Simulated annealing algorithm starts with a relatively high temperature. As temperature drops, it looks for global optimal value of the target function in the solution space with the randomness of possibility, which means it can "jump out of" regional optimal value with a certain possibility and approximate the global optimal value. Simulated annealing algorithm is a universal optimizing algorithm and used widely in engineering, such as production scheduling, construction controlling, machine learning, neural network and signal processing.

In this problem, we first use improved greedy algorithm to obtain the upper bound of the optimal value. The lower bound of the optimal value can be obtained by $\left(\left[\frac{\text{the total area of the uncovered reigion}}{\text{the area of a circle}}\right]+1\right)$. Then we use dichotomy to decrease the range between the upper bound and lower bound. As for a possible solution K, we use simulated annealing algorithm to test if K circles can cover all the uncovered points. Finally, we can obtain the approximate optimal solution.

The application of simulated annealing algorithm:

(1) calculate upperbound and lowerbound.

(2) if lowerbound $< upperbound$, then $\mathrm{K}=\frac{\text{upperbound+lowerbound}}{2}$, then turn to step (10).

(3) simulated annealing initialization: obtain target number K, initial temperature $\mathrm{T}=100000$, minimum temperature $\mathrm{T_{min}}=0.0001$, cooling rate of annealing $\text{coolingrate}=0.003$, initial solution S(produce K random points in the uncovered region as circle centers), calculate the initial energy of initial solution energy(that is

the number of points uncovered) 。

(4) when $T > T_{min}$ and $energy > 0$, do step(5) to step (8).

(5) generate a new solution S'(the method is finding an uncovered point randomly and moving the closest circle nearby so that the point can be covered) [8]

(6) calculate the number of uncovered points $newenergy$ under the solution S'.

(7) if $newenergy < energy$, then accept S' as new solution，$energy = newenergy$，$S = S'$, else accept S' as new solution with the possibility of $e^{\frac{energy-newenergy}{T}}$.

(8) $T = T * (1 - coolingrate)$，turn to step (4).

(9) if $energy = 0$，record the current solution S，$C = S$，$upperbound = k$，

else $lowerbound = k + 1$, turn to step(2)

(10) print solution set C

The MATLAB program is in Appendix 9

Again we take circles with radius of 2 covering a 10×10 square as an example

The first time:

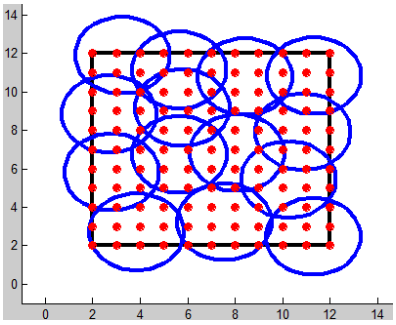


Figure 3-6

running process of the program

lowerbound	*upperbound*	*K*	Final *energy* (the number of uncovered points)
8	16	12	3
13	16	14	0
13	14	13	0

Table 3-1

The solution is K=13

The second time:

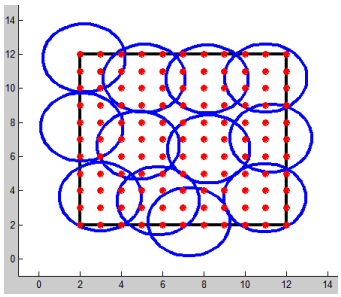


Figure 3-7

running process of the program

$lowerbound$	$upperbound$	K	Final $energy$ (the number of uncovered points)
8	16	12	0
8	12	10	4
11	12	11	3

Table 3-2

The solution is K=12

The third time:

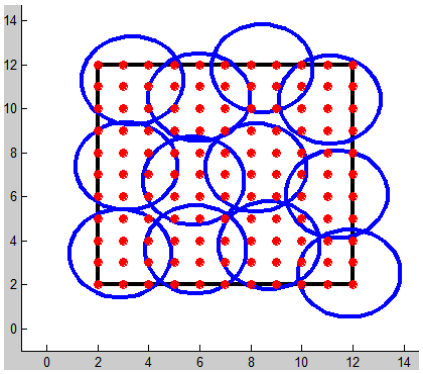


Figure 3-8

running process of the program

$lowerbound$	$upperbound$	K	Final $energy$ (the number of uncovered points)
8	16	12	0
8	12	10	4
11	12	11	5

Table 3-3

The solution is K=12

Chapter IV Case Analysis: The Coverage of Nanjing City

We used base stations with the radius of 10km to cover Nanjing, ignoring the influence of residential place and topography and considering the whole region of Nanjing. In the actual operation, we simplify Nanjing to a JPG format map with resolution of 320×580, scaled down according to the real size. The distance between two nearby pixel points on the map represents $\frac{1}{3}$km in reality，which means the radius of base station coverage on the map equals 30 pixel distance.

4.1 Covering with Specific Bounded Graphs

The first method is to use the method to cover specific bounded graph in Chapter II to cover Nanjing. The concrete realization is as follows: because Nanjing is irregular, we firstly use a single specific bounded graph to include all the regions of Nanjing, And use the theorems in Chapter II to cover this specific bounded graph，then delete all the circles outside Nanjing， and cover the uncovered region using greedy algorithm. Taken this specific bounded graph as an example, like Figure 4-1include Nanjing in a rectangle and cover it by the method mentioned above(result showed in Figure 4-2). 77 circles are used.

Program is in Appendix 10.

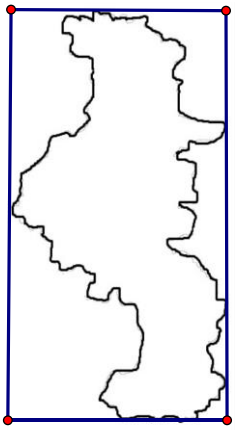


Figure 4-1

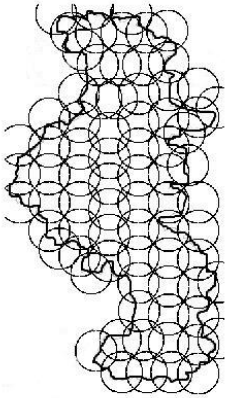


Figure 4-2

4.2 Covering with Simulated Annealing Algorithm

The second method is to use 3.2 simulated annealing algorithm to design base station distribution in Nanjing. First discretize Nanjing with the interval of 10 pixel distance(like Graph 4-3), then apply simulated annealing algorithm，and finally obtain the coverage like Graph 4-4. 55 circles are used in total.

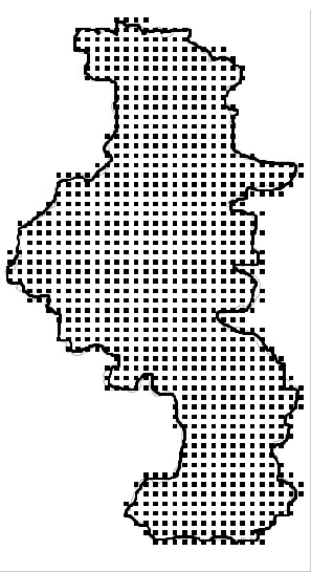


Figure 4-3

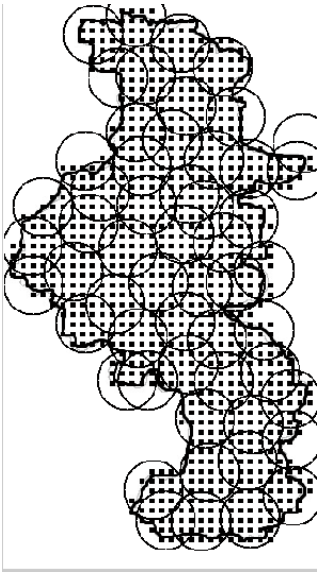


Figure 4-4

4.3 Covering with the Model of Infinite Plane

The method is to use hexagonal-honeycomb way to cover Nanjing. There are 62 circles in total. The result is like Figure 4-5.

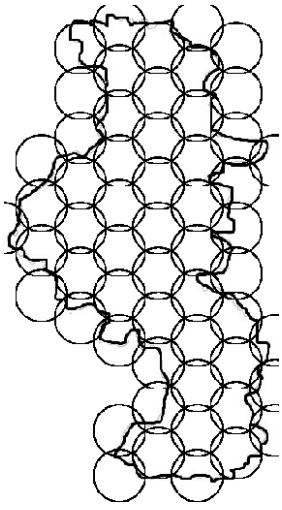


Figure 4-5

4.4 Comparison between Methods

In the analysis from the previous chapter, we conclude that when the radius of base station is 10km and interval is 10 pixel distance, full-plane coverage needs 62 circles and simulated algorithm needs 55 circles. Although using the latter way needs fewer circles, this method does not cover the city completely because the map has been discretized. On the other hand, full-plane model covers Nanjing fully. Moreover, the number of circles needed by simulated annealing algorithm is depend on interval between discrete points--theoretically, the bigger the interval is, the fewer circles are needed.

Finally, like Graph 4-6, when interval is 8 pixel distance, simulated algorithm requires 61 circles, just one fewer than those needed by full-plane coverage. In conclusion, considering real users' demand, two ways of coverage both have their own advantages.

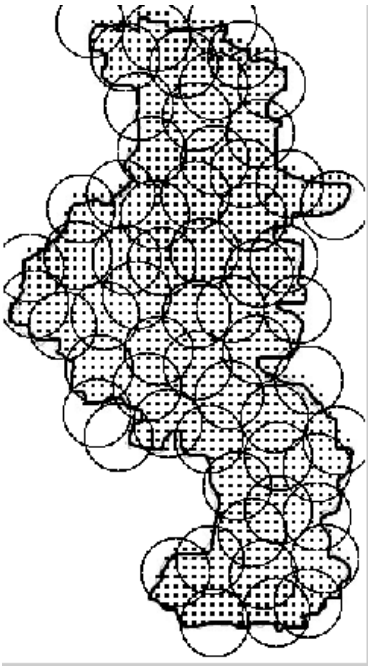
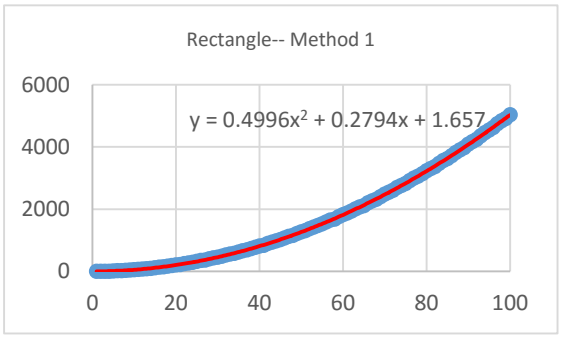


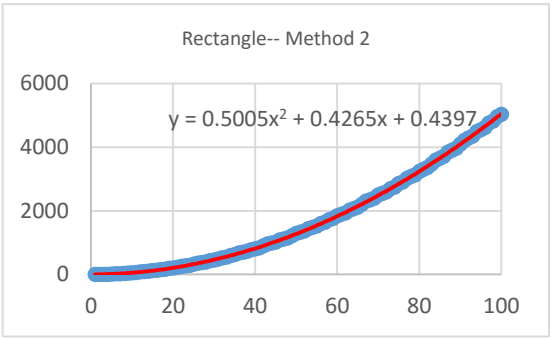
Figure 4-6

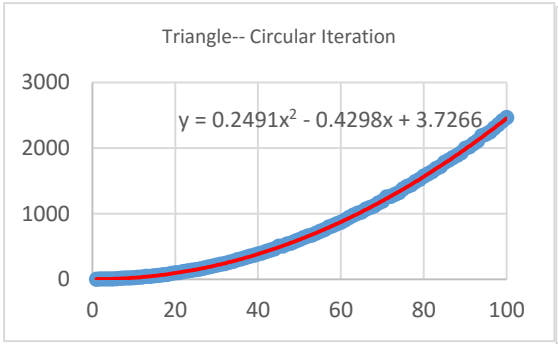
Chapter V Conclusion and Expectation

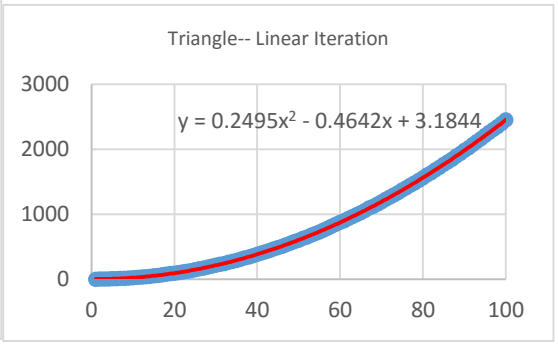
In this paper, we use methods of covering specific bounded graphs, simulated annealing algorithm and the coverage pattern of infinite plane to give general methods to solve the problem of distribution of base stations in Nanjing, and in any cities or areas.

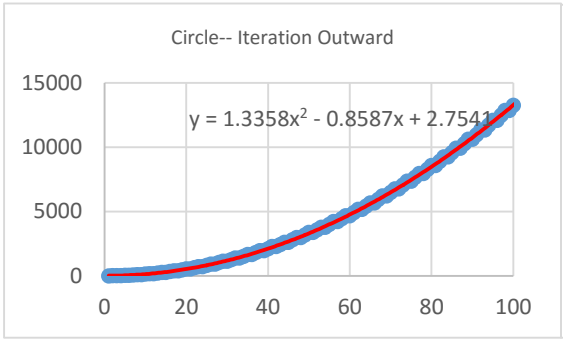
In terms of the coverage methods in Chapter II, we input integers from 1 to 100 to the programs and output the needed number of circles. Graph the following charts:

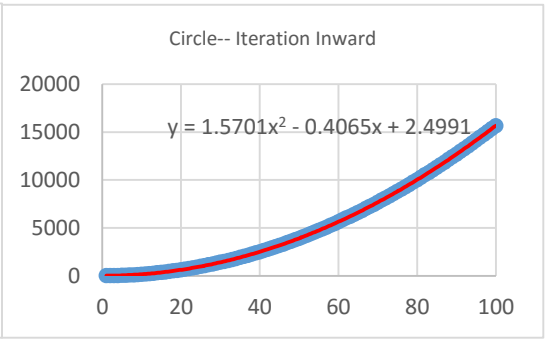












From the figures above, we have a hypothesis:

In terms of coverage any specific bounded graph, along with the increasing of radius of side length, d, the number of unit circles needed to cover $N(d)$ increases in the pattern of Quadratic Increasing, which is $O(d^2) = \lim_{d\to\infty} \frac{N(d)}{d^2} = Constant.$

There are still a lot to discover in this research. Firstly, we use Greedy Algorithm

to generate discrete dots, but in terms of other means to generate discrete dots, will they achieve a better result? Secondly, Greedy Algorithm used in this paper is random. Can we optimize the Greedy Algorithm to both consider the partial and the general? Third, we use three plans to cover the City of Nanjing, can we use them on other cities or specific areas? Last but not least, the coverage radius of base station we adopt is 10 kilometers so that we can use fewer stations. In terms of smaller stations like those of radius of 1 kilometers, how can we change our programs and algorithms to solve the problem better?

Acknowledgement

We are here to give our sincere gratitude to Mr. Tong Zhengqing for his instructions and his leading role to let us think. We thank Prof. Zhou Guofei for his pointing the way for us to study and to research of this paper. And we have to thank our parents for giving us support and care.

Reference

[1]Hans Melissen,Aladar Heppes. Covering a Rectangle with Equal Circle,Periodica Mathematica Hungarica Vol.34 (1-2),(1997) 65-81.

[2]J.B.M.Melissen, Densest packing of congruent circles in an equilateral triangle,Amer.Math.Monthly.100(1993),916-925.

[3]J.B.M.Melissen, Optimal packings of eleven equal circles in an equilateral triangle, Acta Math. Hung. 65 (1994) 389-393.

[4]J.B.M.Melissen and P.C.Schuur,Covering a rectangle with six and seven circles,ElectronicJ.Combin.3(1996),R32,8pp.

[5]M. Mollard and C. Payan, Some progress in the packing of equal circles in a square, Discrete Math. 84 (1990) 303-307.

[6]Bing Gu. The research of optimal coverage of regular region in WSN [J]. The technology and development of computer,2013,(1).

[7] Bing Wu, Ting Zhang, The regional coverage of radio signal transmission [J]. Lianyungang vocational technical institute,2013,(4).

[8]Xiang Hong Yang. The research of optimal facility location coverage based on GIS [E].Beijing: Tsinghua University,2013.

Appendix

Appendix 1, C++

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
	double r[100000];



	cin >> r[1]; int min = 1000000;

	for (double delta = 0; delta<1; delta += 0.000001)
	{
		int n = 1;
		while (r[n] > 1)
		{
			double b = M_PI / ceil(ceil(M_PI / acos(1 - 1 / (2 * r[n] * r[n])))*(1 + delta));
			r[n + 1] = 2 * r[n] * cos(b)*cos(b) - 2 * cos(b)*sqrt(1 - r[n] * r[n] * sin(b)*sin(b)) -
r[n];
			n++;
		}
		int s = 0;
		if (r[n] > 0) s = 1;
		n--;

		for (int i = 1; i <= n; i++)
		{
			s += ceil(ceil(M_PI / acos(1 - 1 / (2 * r[i] * r[i]))*(1 + delta)));

		}
		if (s<min) min = s;
	}
	cout << endl << "T=" << min;
```

```
    system("pause");
    return 0;
}
```

Appendix 2, C++

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
    double a[3000], b[3000], r;
    int k;
    cin >> r;
    for (k = 1; k <= 1000; k++)
    {
        a[2 * k - 1] = sqrt(0.75 + (0.5 + 3 * k - 3)*(0.5 + 3 * k - 3));
        a[2 * k] = 3 * k - 1;
        b[k] = 3 * k*k - 3 * k + 1;
    }
    for (k = 1001; k <= 2000; k++)    b[k] = 3 * k*k - 3 * k + 1;
    int i = 1;
    while (r>a[i] && i<2000) i++;
    cout << b[i];
    system("pause");
    return 0;
}
```

Appendix 3, C++

```
#include<iostream>
#include<cmath>
using namespace std;
double M_PI = 3.1415926535;
```

```
int main()
{
	double a[100000];

	cin >> a[1];

	int n = 1;
	while (a[n] > sqrt(3))
	{
		double b = acos((a[n] - 2 * sqrt(3)) / (2 * ceil((a[n] - 2 * sqrt(3)) / sqrt(3))));
		a[n + 1] = a[n] - 4 * sqrt(3)*sin(b);
		n++;
	}
	int s = 0;
	if (a[n] > 0) s = 1;
	n--;

	for (int i = 1; i <= n; i++)
	{
		s += ceil((a[i] - 2 * sqrt(3)) / sqrt(3)) * 3 + 3;
		cout << ceil((a[i] - 2 * sqrt(3)) / sqrt(3)) * 3 + 3 << ' ';
	}

	cout << endl << "N=" << s;

	system("pause");
	return 0;
}
```

Appendix 4, C++

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
```

```
{
	double a[100000];


	cin >> a[1];

	int n = 1;
	while (a[n] > 2*sqrt(3) )
	{
		double b = acos((a[n] - 2 * sqrt(3)) / (2 * ceil((a[n] - 2 * sqrt(3)) / sqrt(3))));
		a[n + 1] = a[n] - 4 / sqrt(3)*sin(b);
		n++;
	}
	int s = 0;
	if (a[n] > 0 && a[n] <= sqrt(3)) s = 1;
	else if (a[n] > sqrt(3) && a[n] <= 2) s = 2;
	else if (a[n] > 2 && a[n] <= 2 * sqrt(3)) s = 3;
	n--;

	for (int i = 1; i <= n; i++)
	{
		s += (ceil((a[i] - 2 * sqrt(3)) / sqrt(3)) + 2);
		cout << ceil((a[i] - 2 * sqrt(3)) / sqrt(3)) + 2 << ' ';
	}

	cout << endl << "N=" << s;

	system("pause");
	return 0;
}
```

Appendix 5, C++

```
#include<iostream>
#include<cmath>
```

```
using namespace std;
int main()
{
	double a[100], b[100], N[100], alpha[100], beta[100];
	int n = 1;
	cin >> a[1] >> b[1];

	N[n] = (ceil(a[n] / sqrt(2) - 2) + ceil(b[n] / sqrt(2) - 2)) * 2 + 4;
	while (a[n] >= 2 * sqrt(2) && b[n] >= 2 * sqrt(2))
	{		alpha[n] = acos((a[n] - 2 * sqrt(2)) / (2 * ceil(a[n] / sqrt(2) - 2)));
	beta[n] = acos((b[n] - 2 * sqrt(2)) / (2 * ceil(b[n] / sqrt(2) - 2)));
	a[n + 1] = a[n] - 2 * (sin(beta[n]) + cos(beta[n]))*(cos(beta[n]) + 1 / sqrt(2)) / (sqrt(2) +
cos(beta[n]) - sin(beta[n]));
	b[n + 1] = b[n] - 2 * (sin(alpha[n]) + cos(alpha[n]))*(cos(alpha[n]) + 1 / sqrt(2)) / (sqrt(2) +
cos(alpha[n]) - sin(alpha[n]));

	N[n + 1] = (ceil(a[n + 1] / sqrt(2) - 2) + ceil(b[n + 1] / sqrt(2) - 2)) * 2 + 4;

	n = n + 1;
	}
	int s = 0;
	for (int k = 1; k <= n - 1;k=k+1)
		s = s + N[k];
	if (a[1] <= 2 * sqrt(2) || b[1] <= 2 * sqrt(2))
		s = ceil(a[n] / sqrt(2))*ceil(b[n] / sqrt(2));
	else
		s = s+ceil(a[n] / sqrt(2))*ceil(b[n] / sqrt(2));

	cout << s;

		system("pause");
	return 0;
}
```

Appendix 6, C++

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
	double a[100], b[100], N[100], alpha[100], beta[100];
	int n = 1;
	cin >> a[1] >> b[1];

	N[n] = (ceil((a[n] - 2 * sqrt(2) )/ 2) + ceil((b[n] - 2* sqrt(2) )/ 2)) * 2 + 4;
	while (a[n] >= 2 * sqrt(2) && b[n] >= 2 * sqrt(2))
	{
		alpha[n] = acos((a[n] - 2 * sqrt(2)) / (2 * ceil((a[n] - 2 * sqrt(2)) / 2)));
		beta[n] = acos((b[n] - 2 * sqrt(2)) / (2 * ceil((b[n] - 2 * sqrt(2)) / 2)));
		a[n + 1] = a[n] - 2 * (sin(beta[n]) + cos(beta[n]))*(cos(beta[n]) + 1 / sqrt(2)) / (sqrt(2) +
cos(beta[n]) - sin(beta[n]));
		b[n + 1] = b[n] - 2 * (sin(alpha[n]) + cos(alpha[n]))*(cos(alpha[n]) + 1 / sqrt(2)) / (sqrt(2)
+ cos(alpha[n]) - sin(alpha[n]));
		N[n + 1] = (ceil((a[n + 1] - 2 * sqrt(2)) / 2) + ceil((b[n + 1] - 2 * sqrt(2)) / 2)) * 2 + 4;

		n = n + 1;
	}
	int s = 0;
	for (int k = 1; k <= n - 1; k = k + 1)
		s = s + N[k];
	if (a[1] <= 2 * sqrt(2) || b[1] <= 2 * sqrt(2))
		s = ceil(a[n] / sqrt(2))*ceil(b[n] / sqrt(2));
	else
		s = s + ceil(a[n] / sqrt(2))*ceil(b[n] / sqrt(2));

	cout << endl << "N=" << s;
	system("pause");
	return 0;
}
```

Appendix 7, C++

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{double b,a1,beta;
 cin>>b>>a1;
 beta=acos((b-2*sqrt(2))/(2*ceil(b/sqrt(2)-2)));
 double n=ceil(a1*(sqrt(2)+cos(beta)-sin(beta))/((cos(beta)+sin(beta))*(cos(beta)+sqrt(2)/2)));
 cout<<n*ceil(b/sqrt(2));
 system("pause");
 return 0;
}
```

Appendix 8, MATLAB

```
n=10;r=2;delete_rate=0.7;kmax=100;

hold on
for i=1:n+1
    for j=1:n+1
        square(i,j)=0;
    end
end
f=0;cnt=0;k=0;
while(k<kmax)
    max=0;i1=0;j1=0;
    for i=1:n+1
        for j=1:n+1
            s=0;
            if square(i,j)==0
               for p=i-r:i+r
                  for q=j-r:j+r
                      if
p>0&p<=n+1&q>0&q<=n+1&sqrt((i-p)^2+(j-q)^2)<=r&square(p,q)==0
```

```
                        s=s+1;
                    end
                end
            end
        end
        if s>max
            max=s;i1=i;j1=j;
        end
    end
end
cnt=cnt+1;site(cnt,1)=i1;site(cnt,2)=j1;
        for p=i1-r:i1+r
            for q=j1-r:j1+r
                if p>=1&p<=n+1&q>=1&q<=n+1&sqrt((i1-p)^2+(j1-q)^2)<=r
                    square(p,q)=1;
                end
            end
        end
        f=1;
        for i=1:n+1
            for j=1:n+1
                if square(i,j)==0
                    f=0;
                end
            end
        end
if f==1
    k=k+1;
    if k==1 || cnt<ans
        ans=cnt;ans
        for i=1:cnt
            a(i,1)=site(i,1);a(i,2)=site(i,2);
        end
    end
    for i=1:cnt*delete_rate
```

```
            x=round(rand*(cnt-1))+1;
            for j=x:cnt-1
                site(j,1)=site(j+1,1);site(j,2)=site(j+1,2);
            end
            cnt=cnt-1;
        end
        for i=1:n+1
            for j=1:n+1
                square(i,j)=0;
            end
        end

        for i=1:cnt
            for p=site(i,1)-r:site(i,1)+r
                for q=site(i,2)-r:site(i,2)+r
                    if p>=1&p<=n+1&q>=1&q<=n+1&sqrt((site(i,1)-p)^2+(site(i,2)-q)^2)<=r
                        square(p,q)=1;f=0;
                    end
                end
            end
        end
    end
end

rectangle('Position',[2,2,n,n],'Linewidth',3)
for i=1:ans
    alpha=0:pi/20:2*pi;
    x=r*cos(alpha);
    y=r*sin(alpha);
    plot(x+a(i,1)+1,y+a(i,2)+1,'Linewidth',3);
end
for i=2:2+n
    for j=2:2+n
        plot(i,j,'ro','MarkerFaceColor','r');
    end
```

```
end
axis([-1 20 -1 20])
```

Appendix 9, MATLAB

```
wide=10;high=10;r=2;

hold on
for i=1:high+1
    for j=1:wide+1
        square(i,j)=0;
    end
end
%calculate the upper bound by greedy algorithm
f=0;upperbound=0;
while(f==0)
    max=0;i1=0;j1=0;
    for i=1:high+1
        for j=1:wide+1
            s=0;
            if square(i,j)==0
               for p=i-r:i+r
                  for q=j-r:j+r
                      if
p>0&p<=high+1&q>0&q<=wide+1&sqrt((i-p)^2+(j-q)^2)<=r&square(p,q)==0
                          s=s+1;
                      end
                  end
               end
            end
            if s>max
                max=s;i1=i;j1=j;
            end
        end
    end
```

```
        upperbound=upperbound+1;res(upperbound,1)=i1;res(upperbound,2)=j1;
                for p=i1-r:i1+r
                    for q=j1-r:j1+r
                        if p>=1&p<=high+1&q>=1&q<=wide+1&sqrt((i1-p)^2+(j1-q)^2)<=r
                            square(p,q)=1;
                        end
                    end
                end
                f=1;
                for i=1:high+1
                    for j=1:wide+1
                        if square(i,j)==0
                            f=0;
                        end
                    end
                end
end
ans=upperbound;
lowerbound=floor(high*wide/(pi*r^2))+1;    %calculate the lower bound

 % narrow the range by dichotomy
while(lowerbound<upperbound)
    k=floor((lowerbound+upperbound)/2);
    t=100000;tmin=0.0001;coolingrate=0.003;energy=0;
    for i=1:high+1
        for j=1:wide+1
            square(i,j)=0;
        end
    end
    %generate the initial solution
    for i=1:k
        posi(i,1)=0;posi(i,2)=0;f=0;p=0;
        while(posi(i,1)<1 || posi(i,1)>high+1 || posi(i,2)<1 || posi(i,2)>wide+1 ||
f==0&&p<100000)
            posi(i,1)=floor(rand*high)+1;posi(i,2)=floor(rand*wide)+1;
```

```
                f=1;p=p+1;
                for j=1:i-1
                    if sqrt((posi(i,1)-posi(j,1))^2+(posi(i,2)-posi(j,2))^2)<1.5*r
                        f=0;
                    end
                end
            end
            for p=floor(posi(i,1)-r):posi(i,1)+r
                for q=floor(posi(i,2)-r):posi(i,2)+r
                    if
p>=1&p<=high+1&q>=1&q<=wide+1&sqrt((posi(i,1)-p)^2+(posi(i,2)-q)^2)<=r
                        square(p,q)=1;
                    end
                end
            end
        end
        for i=1:high+1
            for j=1:wide+1
                if square(i,j)==0
                    energy=energy+1;
                end
            end
        end

        %simulated annealing
        while(t>tmin&energy>0)
            change=floor(rand*(energy-1))+1;
            p=0;
            for i=1:high+1
                for j=1:wide+1
                    if square(i,j)==0
                        p=p+1;
                        if (p==change)
                            i1=i;j1=j;break;
                        end
```

```
            end
        end
        if (p==change) break;end
    end
    min=100000;minj=0;
    for i=1:k
        if sqrt((posi(i,1)-i1)^2+(posi(i,2)-j1)^2)<min
            min=sqrt((posi(i,1)-i1)^2+(posi(i,2)-j1)^2);minj=i;
        end
    end
    ti=posi(minj,1);tj=posi(minj,2);
    posi(minj,1)=0;posi(minj,2)=0;
    while(posi(minj,1)<1 || posi(minj,1)>high+1 || posi(minj,2)<1 || posi(minj,2)>wide+1)
      alpha=rand*2*pi;dist=rand*r+0.01;
      posi(minj,1)=i1+dist*cos(alpha);posi(minj,2)=j1+dist*sin(alpha);
    end
    for i=1:high+1
        for j=1:wide+1
          square(i,j)=0;
        end
    end
    for i=1:k
       for p=floor(posi(i,1)-r):posi(i,1)+r
           for q=floor(posi(i,2)-r):posi(i,2)+r
               if
p>=1&p<=high+1&q>=1&q<=wide+1&sqrt((posi(i,1)-p)^2+(posi(i,2)-q)^2)<=r
                  square(p,q)=1;
               end
           end
       end
    end
    newenergy=0;
    for i=1:high+1
        for j=1:wide+1
          if square(i,j)==0
```

```
                    newenergy=newenergy+1;
                end
            end
        end
    % jump out of the regional optimal value with a certain possibility
        if (newenergy<energy || (newenergy>=energy&rand<exp((energy-newenergy)/t)))
            energy=newenergy;

            if (energy==0)
                ans=k;
                for i=1:k
                    res(i,1)=posi(i,1);res(i,2)=posi(i,2);
                end

            end
        else
            posi(minj,1)=ti;posi(minj,2)=tj;
        end
        t=t*(1-coolingrate);

    end
    if (energy==0)
        upperbound=k;
    else
        lowerbound=k+1;
    end

end
%print out
rectangle('Position',[2,2,wide,high],'Linewidth',3);
for i=1:ans
    alpha=0:pi/20:2*pi;
    x=r*cos(alpha);
    y=r*sin(alpha);
    plot(y+res(i,2)+1,x+res(i,1)+1,'Linewidth',3);
```

```
end
for i=2:2+n
    for j=2:2+n
        plot(i,j,'ro','MarkerFaceColor','r');
    end
end
axis([-1 20 -1 20])
```

Appendix 10, MATLAB

```
A=imread('D:\32.jpg'); %input the image
A=rgb2gray(A);

[n m]=size(A);
for i=1:n
    for j=1:m
        A1(i,j)=0;
    end
end
imin=n;imax=0;jmin=m;jmax=0;cnt=0;
for i=1:n
    p=1;
    while(A(i,p)==255 & p<m) p=p+1;end
    if (p<m)
        q=m;imax=i;if i<imin    imin=i;end
        while(A(i,q)==255 & q>p) q=q-1;end
        if p<jmin jmin=p;end
        if q>jmax jmax=q;end
        for j=p:q
            A1(i,j)=1;
        end
    end
end

n=1;cnt=0;
a(1)=541;b(1)=302;r=30;
while (a(n) >= 2*sqrt(2)*r & b(n) >= 2*sqrt(2)*r)
    alpha(n) = acos((a(n) - 2 * r * sqrt(2)) / (2 *r* ceil(a(n) / sqrt(2)/r - 2)));
    beta(n) = acos((b(n) - 2 * r * sqrt(2)) / (2 *r* ceil(b(n) / sqrt(2)/r - 2)));
    a(n + 1) = a(n) -
2*r*(cos(beta(n))+sin(beta(n)))*(cos(beta(n))+sqrt(2)/2)/(sqrt(2)+cos(beta(n))-sin(beta(n)));
    b(n + 1) = b(n) -2*r*
```

```
(cos(alpha(n))+sin(alpha(n)))*(cos(alpha(n))+sqrt(2)/2)/(sqrt(2)+cos(alpha(n))-sin(alpha(n)));
    N(n + 1) = (ceil(a(n + 1) / sqrt(2)/r - 2) + ceil(b(n + 1) / sqrt(2)/r - 2)) * 2 + 4;
    T=0;
    for i=1:n-1

T=T+r*(cos(alpha(i))+sin(alpha(i)))*(cos(alpha(i))+sqrt(2)/2)/(sqrt(2)+cos(alpha(i))-sin(alpha(i)))
;
    end
    S=0;
    for i=1:n-1

S=S+r*(cos(beta(i))+sin(beta(i)))*(cos(beta(i))+sqrt(2)/2)/(sqrt(2)+cos(beta(i))-sin(beta(i)));
    end

    cnt=cnt+1;
    circle(cnt,1)=T+r*sqrt(2)/2;circle(cnt,2)=a(n)-r*sqrt(2)/2+S;
    cnt=cnt+1;
    circle(cnt,1)=T+r*sqrt(2)/2;circle(cnt,2)=S+r*sqrt(2)/2;
    cnt=cnt+1;
    circle(cnt,1)=T+b(n)-r*sqrt(2)/2;circle(cnt,2)=a(n)-r*sqrt(2)/2+S;
    cnt=cnt+1;
    circle(cnt,1)=T+b(n)-r*sqrt(2)/2;circle(cnt,2)=S+r*sqrt(2)/2;

    for p=1:ceil(b(n)/sqrt(2)/r-2)
        cnt=cnt+1;
        circle(cnt,1)=T+r*sqrt(2)+r*(2*p-1)*cos(beta(n));circle(cnt,2)=a(n)-r*sin(beta(n))+S;
        cnt=cnt+1;
        circle(cnt,1)=T+r*sqrt(2)+r*(2*p-1)*cos(beta(n));circle(cnt,2)=r*sin(beta(n))+S;
    end
    for q=1:ceil(a(n)/sqrt(2)/r-2)
        cnt=cnt+1;
        circle(cnt,1)=T+b(n)-r*sin(alpha(n));circle(cnt,2)=r*sqrt(2)+r*(2*q-1)*cos(alpha(n))+S;
        cnt=cnt+1;
        circle(cnt,1)=T+r*sin(alpha(n));circle(cnt,2)=r*sqrt(2)+r*(2*q-1)*cos(alpha(n))+S;
    end
     n=n+1;
end
 T=0;
    for i=1:n-1

T=T+r*(cos(alpha(i))+sin(alpha(i)))*(cos(alpha(i))+sqrt(2)/2)/(sqrt(2)+cos(alpha(i))-sin(alpha(i)))
;
    end
    S=0;
```

```
    for i=1:n-1

S=S+r*(cos(beta(i))+sin(beta(i)))*(cos(beta(i))+sqrt(2)/2)/(sqrt(2)+cos(beta(i))-sin(beta(i)));
    end
if a(n)<=r*2*sqrt(2)
    if a(n)>r*sqrt(2)
        for p=1:ceil(b(n)/sqrt(2)/r);
            cnt=cnt+1;

circle(cnt,1)=T+(b(n)/(ceil(b(n)/sqrt(2)/r)))*(2*p-1)/2;circle(cnt,2)=sqrt(1-(b(n)/(ceil(b(n)/sqrt(2)/r
)))*(b(n)/(ceil(b(n)/sqrt(2)/r)))/4)+S;
            cnt=cnt+1;

circle(cnt,1)=T+(b(n)/(ceil(b(n)/sqrt(2)/r)))*(2*p-1)/2;circle(cnt,2)=a(n)-sqrt(1-(b(n)/(ceil(b(n)/sqr
t(2)/r)))*(b(n)/(ceil(b(n)/sqrt(2)/r)))/4)+S;
        end

    else
        for p=1:ceil(b(n)/sqrt(2)/r);
            cnt=cnt+1;
            circle(cnt,1)=T+(b(n)/(ceil(b(n)/sqrt(2)/r)))*(2*p-1)/2;circle(cnt,2)=a(1)/2;
        end
    end
else
    if b(n)>r*sqrt(2)
        for p=1:ceil(a(n)/sqrt(2)/r);
            cnt=cnt+1;

circle(cnt,2)=T+(a(n)/(ceil(a(n)/sqrt(2)/r)))*(2*p-1)/2;circle(cnt,1)=sqrt(1-(a(n)/(ceil(a(n)/sqrt(2)/r
)))*(a(n)/(ceil(a(n)/sqrt(2)/r)))/4)+S;
            cnt=cnt+1;

circle(cnt,2)=T+(a(n)/(ceil(a(n)/sqrt(2)/r)))*(2*p-1)/2;circle(cnt,1)=b(n)-sqrt(1-(a(n)/(ceil(a(n)/sqrt
(2)/r)))*(a(n)/(ceil(a(n)/sqrt(2)/r)))/4)+S;
        end

    else
        for p=1:ceil(a(n)/sqrt(2)/r);
            cnt=cnt+1;
            circle(cnt,2)=T+(a(n)/(ceil(a(n)/sqrt(2)/r)))*(2*p-1)/2;circle(cnt,1)=b(1)/2;
        end
    end
end
[n m]=size(A);r=30;
```

```
for i=1:cnt
    p1=imax-floor(circle(i,2))+1;
    q1=floor((circle(i,1)+1))+jmin;
    circle(i,1)=p1;circle(i,2)=q1;

end

for i=1:cnt

    if (A1(circle(i,1),circle(i,2))==0)

        for j=i:cnt-1
            circle(j,1)=circle(j+1,1);
            circle(j,2)=circle(j+1,2);
        end
        cnt=cnt-1;
        i=i-1;
    end
end

res=cnt;
cnt=0;
for i=imin:imax
    for j=jmin:jmax
      if (A1(i,j)==1)
            f=0;
            for p=1:res
                if sqrt((circle(p,1)-i)^2+(circle(p,2)-j)^2)<=r
                    f=1;break;
                end
            end

            if f==0 A1(i,j)=2;cnt=cnt+1;posi(cnt,1)=i;posi(cnt,2)=j;end
      end
    end
end

f=0;
if cnt==0 f=1;end
while(f==0)
    max=0;i1=0;
    for i=1:cnt

        s=0;
```

```
            if A1(posi(i,1),posi(i,2))==2
                for p=posi(i,1)-r:posi(i,1)+r
                    for q=posi(i,2)-r:posi(i,2)+r
                        if
p>imin&p<=imax&q>jmin&q<=jmax&sqrt((posi(i,1)-p)^2+(posi(i,2)-q)^2)<=r&A1(p,q)==2
                            s=s+1;
                        end
                    end
                end

                if s>max
                    max=s;i1=i;
                end
            end
        end
        res=res+1;circle(res,1)=posi(i1,1);circle(res,2)=posi(i1,2);
                for p=posi(i1,1)-r:posi(i1,1)+r
                    for q=posi(i1,2)-r:posi(i1,2)+r
                        if
p>=imin&p<=imax&q>=jmin&q<=jmax&sqrt((posi(i1,1)-p)^2+(posi(i1,2)-q)^2)<=r
                            A1(p,q)=1;
                        end
                    end
                end
                f=1;
                for i=1:cnt

                    if A1(posi(i,1),posi(i,2))==2
                        f=0;break;
                    end

                end
end
for i=1:res

        for p=floor(circle(i,1)-r):circle(i,1)+r
            for q=floor(circle(i,2)-r):circle(i,2)+r
                if p>1&p<n&q>1&q<m&abs(sqrt((p-circle(i,1))^2+(q-circle(i,2))^2)-r)<1
                    A(p,q)=0;
                end
            end
        end
end
disp(res);
```

```
imshow(A);
```