Construction of Optimal Tubular Networks in Arbitrary Regions in $\mathbf{R}^3$

by

Wenzhe Jiang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Applied Mathematics

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this thesis, we describe various algorithms for the construction of tubular networks in an arbitrary three-dimensional region that possesses a principal direction along which the cross-section varies. The region can be digitized into a series of blocks B_i, each of which exhibits no variation in the principle direction. Tubular networks with one inlet and one outlet are constructed by connecting the series of blocks packed with parallel cylindrical tubes.

Packing tubes into a block B_i can be simplified to packing circles into its cross section C_i which is approximated by a polygon. Building upon the basic method from [12], a set of novel circle-packing algorithms are developed to possess the following desirable features. Firstly, circles are first packed into an interior region which is common to several or all blocks and the remainder of the unpacked region are packed afterwards. Secondly, larger circles are placed primarily in the central part of the region. Lastly, at a certain stage of circle-packing, all the packed circles are moved towards the center of mass by a fictitious force so that as much empty space as possible is left along the boundary.

Three fundamental connections are used to construct a tubular network with one inlet and one outlet: (i) endcap connection–two 90-degree bends that connect two adjacent tubes at their ends, (ii) a merge operation in which a tube flows into an adjacent tube, and (iii) a shift operation in which a tube is shifted into an adjacent position. Tubes at the extreme ends of a network must be connected by endcaps and the other tubes can be connected via any one of the above connections. A set of algorithms are developed to generate all the possible solutions of constructing tubular networks and to check the feasibility of solutions to make sure there is no dead end or isolated loop.

Among all the feasible networks, an optimal network should be chosen with respect to a certain measure. Obviously, enumerating all the possible solutions of constructing feasible networks is extremely time-consuming and sometimes it can take millions of years. Therefore, a genetic algorithm with only mutation is applied here to randomly search for the best network. In this genetic algorithm, every fundamental connection could mutate to any other fundamental one. Thus every network could mutate to any other one and genetic algorithm could find the globally best network.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my co-supervisors Dr. Edward Vrscay and Dr. Sean Peterson and a co-investigator in our research group Dr. Franklin Mendivil, for their support, guidance and care. They have provided an exceptional atmosphere for innovating, without which I would not have attained such an achievement. Thanks to them, my problem-solving ability and presentation skill have been significantly enhanced.

I would also like to acknowledge the financial support from the Faculty of Mathematics, University of Waterloo, the Department of Applied Mathematics, University of Waterloo and an NSERC Collaborative Research and Development Grant.

Dedication

To my parents, Hongyu Li and Kiki
for the support, companion and happiness they have brought me.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis, we consider the problem of constructing tubular networks in arbitrary three-dimensional regions. Such a problem has a wide spectrum of industrial applications, e.g., fluid transfer, heat transfer. This particular study has been inspired, in part, by the so-called "redneck barbecue pool heater" [1], which is shown below.

Figure 1.1: The redneck barbecue pool heater

As its name suggests, the barbecue pool heater is designed to heat water in an outdoor pool. Water from the pool is pumped into the barbecue via an inlet pipe. As the water travels through the network inside the barbecue, it is heated. The heated water exits through an outlet pipe and travels to the pool. The two most important elements of the barbecue pool heater are respectively a physically valid network with one inlet and one

outlet and tubes tightly packed in the envelope of the pool heater.

In this thesis, we generalize the barbecue pool heater problem in the following ways:

a). Generalize the envelope to an arbitrary 3D region and pack as many tubes into it as possible.

b). Construct a physically valid network with more branchings.

The following 3D image is an example of a generalized tubular network which is produced by a series of algorithms developed for this problem.

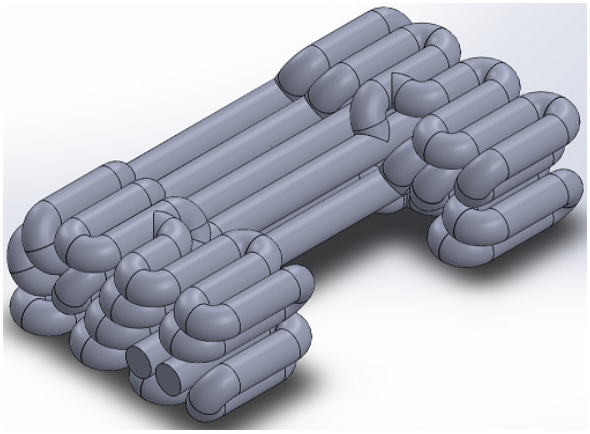


Figure 1.2: An example of the generalized tubular network

As shown in the above figure, this tubular network is constructed for a 3D region with shape similar to a saddle bag. Tubes are tightly packed and there are branchings resulted from merge and shift operations on tubes. Note that in the generalized barbecue pool heater problem, tubes are packed touching each other, which is different from the actual barbecue pool heater. However, one can narrow down the diameters of tubes packed in the constructed tubular network in order to construct an actual pool heater.

In the general case, given a 3D region, we first set a principal direction and discretize the region into several blocks along this direction. Each of these blocks exhibits no variation in cross section along the principal direction. An example is shown in Figure 1.3.

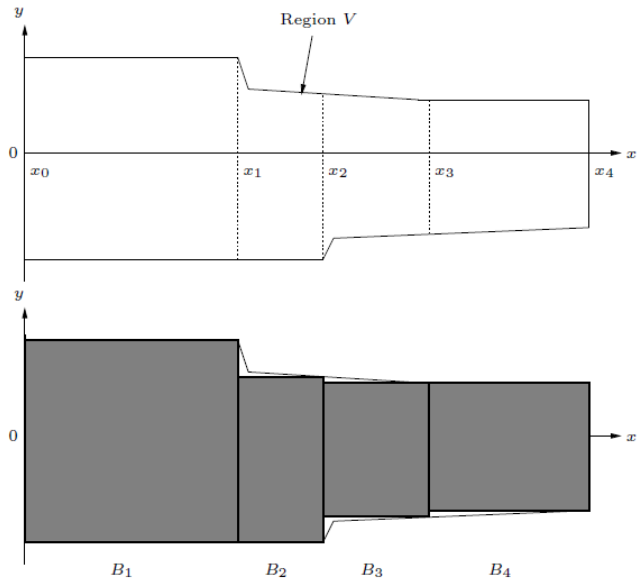


Figure 1.3: Discretizing a 3D region into blocks along its principle direction

Now the problem of constructing tubular networks in the 3D region can be divided into two parts:

Part 1. Pack tubes into each block as compactly as possible.

Part 2. Connect all the tubes packed in the 3D region to construct a feasible tubular network with one inlet and one outlet.

In general, a huge number of solutions to this problem will exist. An optimal solution or solutions can be chosen according to the application being considered. For instance, a solution can be chosen to be optimal in terms of maximum internal volume, or heat flux (if applied to heat transfer problem) or characteristics of the flow of a fluid through the network, etc. More details of the application in the heat transfer problem are to be found in the paper [17].

Because of the assumed longitudinal symmetry, the problem of packing tubes into each block is reduced to the packing of circles into the two-dimensional cross section of the block. This is the well-known problem of circle packing in the plane, an NP-hard combinatorial optimization problem, i.e., no procedure is able to solve it in deterministic polynomial time [15]. The "simplest" cases of packing uniform sized circles inside a square or inside a

circle are provably solved to theoretical optimality only for a few instances (up to tens of circles), in spite of the significant effort spent on variants of the problem in recent decades [5]. Fraser and George [10] focused on the stocking of cylindrical paper-rolls, in other words, putting a number of identical circular bins into a given rectangular box. Dowsland [7] treated the problem from a different perspective, i.e., finding the most suitable box in order to contain a given number of cylindrical objects. Szabó et al. [27] proposed new approaches to circle packing in a square. Wang et al. [16] discussed an improved algorithm for the packing of unequal circles within a larger containing circle. Addis et al. [2] also studied how to efficiently pack unequal disks in a circle. However, very few researchers have worked on the problem of packing different-sized circles into an arbitrary polygon. We investigate such algorithms and more details are to be found in the paper [18].

Connecting tubes packed in the 3D region can be considered as finding a partner or partners for each tube because some tubes can be connected to only one tube while some other tubes can get connected to multiple tubes by various connection operations.

In the situation where a tube is allowed to be connected to only one tube, the connection problem can be converted to one of finding maximum matchings in a graph, i.e., pairing as many nodes in the graph as possible. Numerous studies have been conducted by researchers. The maximum matching problem is one of the most studied problems in the area of graph algorithms. The first polynomial time algorithm to solve this problem was devised by Edmonds in 1965 and runs in time $O(|V|^4)$ [9]. Over the years, many improvements have been made [19]. Micali and Vazirani [23] devised an algorithm for finding maximum matching running in time $O(|E|\sqrt{|V|})$. The algorithm of Mucha and Sankowski [25] runs asymptotically faster on dense graphs–its runtime is $O(|V|^\omega)$ where $\omega < 2.38$ is the exponent needed to perform fast matrix multiplication.

However, our goal is not to find a maximum matching for one pass. Instead, the goal is to enumerate all the maximum matchings of a given graph. We can use the Minimum Degree Matching Algorithm, which is a greedy algorithm that repeatedly connects tubes with the highest priority of getting connected. If combined with Depth-first Search it can enumerate all the maximum matchings efficiently.

In the situation where tubes are allowed to be connected to multiple tubes by various connection operations, we modify the Minimum Degree Matching Algorithm and introduce a new concept of "connectivity value" to enable tubes to be connected to multiple tubes.

Solutions generated by the above algorithms are not guaranteed to be physically valid. Therefore, some invalid solutions must be eliminated algorithmically. A tubular network is physically valid if and only if there is no isolated loop or dead end. An isolated loop is a group of self-connected tubes with no connection with other tubes and a dead end is a group of self-connected tubes with only one path to the other part of the tubular network. These two invalid groups can be algorithmically identified by some graph algorithms for identifying "disconnected graph" and a "bridge" respectively.

Enumerating all the feasible solutions is impractical due to the huge number of solutions. Therefore, we explore a genetic algorithm to efficiently obtain a good or optimal solution. A genetic algorithm (GA) is a search heuristic that mimics the process of natural selection. This metaheuristic is routinely used to generate useful solutions to optimization and search problems. In a genetic algorithm, a population of candidate solutions to an optimization problem is evolved towards better solutions. The evolution usually starts from an initial population of randomly generated individuals (solutions) and is an iterative process, with the population in each iteration called a generation [24]. A new generation is formed by generating new offspring by altering and/or recombining existing individuals that are stochastically selected and eliminating unsatisfying individuals. Commonly, a genetic algorithm terminates when either a given number of generations has been produced, or a satisfactory fitness level has been reached for the population. In our problem, the genetic algorithm has only the mutation operator which generates new individuals and it is capable of altering an existing solution to any other solution in the entire solution space.

The thesis is organized in the following manner. In Chapter 2, we first describe an existing algorithm for packing different-sized circles into a rectangular region [12]. Then we extend this algorithm to packing circles into an arbitrary polygon. We also introduce and describe a set of novel circle-packing algorithms. In Chapter 3 are described various algorithms for building connections based on a circle-packing in the 3D region. We also present two algorithms for verifying the feasibility of each potential solution based on a series of theorems in graph theory. In Chapter 4, we discuss a genetic algorithm with only a mutation operator to generate new individuals.

Chapter 2

Circle-packing algorithms

In this chapter, we describe various algorithms of packing different-sized circles into an arbitrary region approximated by a polygon, maximizing the covered area. In Section 1, we describe an existing algorithm proposed by George, George and Lamar [12] which packs different-sized circles into rectangular regions. For convenience, we shall refer to this algorithm throughout the thesis as the "GGL algorithm". An extension of this algorithm to packing circles into arbitrary polygons is discussed in Section 2. However, this type of algorithm does not meet our specific needs. Therefore, a series of novel circle-packing algorithms are developed in order for the ultimate circle-packing to possess the following features:

1. Larger circles are situated primarily in the interior of the region.
2. As much of the remaining empty space as possible is along the boundary.

2.1 GGL Circle-Packing Algorithm

In this section, we describe the GGL algorithm of packing different-sized circles into a rectangular region. The GGL algorithm was designed for packing pipes of different diameters into a shipping container and by the heuristics of this algorithm it starts packing circles at corners and develops both upwards and to the right with the assistance of sides and two-circle packing. Its basic method of placing circles by position numbers will be used in the algorithms described in the next few sections.

2.1.1 Problem description

The rectangular region in the xy-plane is $R = [0,A] \times [0,B]$, i.e., $0 \leq x \leq A$, $0 \leq y \leq B$, where $A, B > 0$ define the lengths of the sides of the rectangle. The sides of the rectangle are numbered in the counterclockwise direction:

(a) Side 1: the left vertical boundary line, denoted by the variable $s = 1$.

(b) Side 2: the bottom horizontal boundary line, denoted by the variable $s = 2$.

(c) Side 3: the right vertical boundary line, denoted by the variable $s = 3$.

(d) Side 4: the upper horizontal boundary line, denoted by the variable $s = 4$.

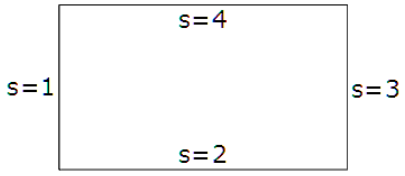


Figure 2.1: Numbering of the sides of a rectangle

A set of N candidate circles are given, each of which has a prescribed radius R_i, $1 \leq i \leq N$. The problem is to pack this set of candidate circles with prescribed radii into a prescribed region and to maximize the area covered by the packed circles.

An "occupancy variable" δ_i is used to denote whether the ith circle is used in the packing:

$$\delta_i = \begin{cases} 1 & \text{if } i\text{th circle is used in the packing} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

If the ith circle is used, i.e., $\delta_i = 1$, then the coordinates of its center are denoted by (x_i, y_i).

The circles employed in the packing must satisfy two sets of constraints:

1. They must be situated inside the rectangle or at most tangent to the boundaries, i.e.,
$$R_i \leq x_i \leq A - R_i, \quad R_i \leq y_i \leq B - R_i \tag{2.2}$$
2. Any pair of circles employed in the packing can intersect at most at one point, i.e., the distance between the centers of the two circles must be greater that or equal to the sum of their radii, i.e.,
$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq R_i + R_j \tag{2.3}$$

2.1.2 Rules and formulas for packing circles

Suppose that the kth circle of radius R_k, $k \geq 1$, is being considered for packing. The three types of possible positions for it are as follows:

1. Placed at the corners touching two sides.

   (a) **Lower left corner.** This position is denoted as $p = 1$. The coordinates of the center of circle k, if packed, are $(x_k, y_k) = (R_k, R_k)$.

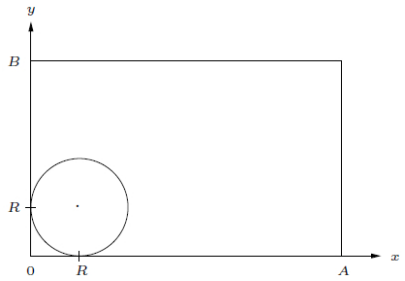


Figure 2.2: Position No. 1, $p = 1$

   (b) **Lower right corner.** This position is denoted as $p = 2$. The coordinates of the center of circle k, if packed, are $(x_k, y_k) = (A - R_k, R_k)$.

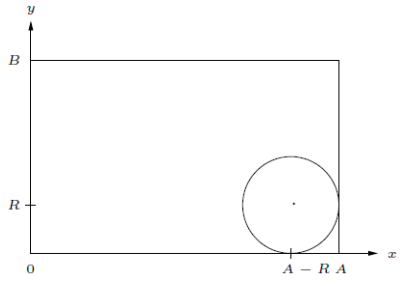


Figure 2.3: Position No. 2, $p = 2$

2. Placed touching one side and one previously packed circle i.

   Consider a circle of radius R_a with center at (x_a, y_a)–this is the circle that is already packed. We now determine the three possible positions of a circle of radius R. Let (x, y) be the coordinates of the center of this circle if it is successfully packed.

(a) Tangent to the ith circle and touching side 1 at one point:

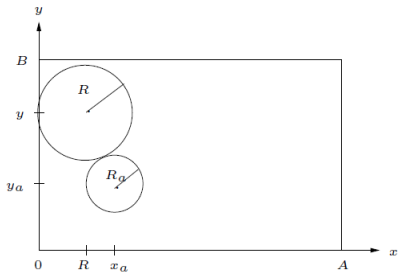


Figure 2.4: Side No. 1

For a solution to exist, the following inequality must be satisfied,

$$x_a \leq 2R + R_a. \tag{2.4}$$

In this case, $x = R$ and there are two solutions for y:

$$y = y_a \pm \sqrt{(R_a + x_a)(2R + R_a - x_a)}. \tag{2.5}$$

Since the GGL algorithm packs circles both upwards and to the right, the solution with negative sign should be discarded.

(b) Tangent to the ith circle and touching side 2 at one point:

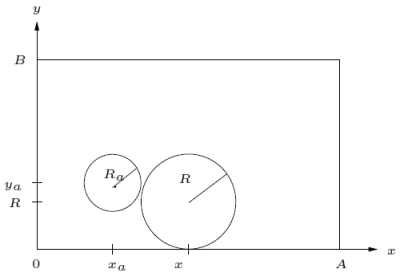


Figure 2.5: Side No. 2

This is a $\pi/2$-rotated version of the previous one. For a solution to exist, it must have that

$$y_a \leq 2R + R_a. \tag{2.6}$$

In this case, $y = R$ and there are two solutions for x:

$$x = x_a \pm \sqrt{(R_a + y_a)(2R + R_a - y_a)} \tag{2.7}$$

The solution with negative sign should be discarded since the packing must proceed to the right by GGL algorithm.

(c) Tangent to the ith circle and touching side 3 at one point:

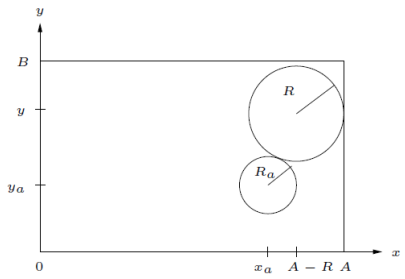


Figure 2.6: Side No. 3

This is a an inverted version of the Side No. 1 Problem. For a solution to exist, we must have

$$x_a \geq A - 2R - R_a. \tag{2.8}$$

In this case, $x = A - R$ and there are two solutions for y

$$y = y_a \pm \sqrt{(R_a + R)^2 - (x_a - (A - R))^2}. \tag{2.9}$$

Again, the solution with negative sign should be discarded since the packing must proceed upwards by GGL algorithm.

Note that only Sides No. 1, 2 and 3 are used for packing, which is a consequence of the physical motivation for the algorithm development of packing pipes into a container. Of course Side No. 4 can be used for packing, which will be discussed in Section 2.2.

3. Placed touching two previously packed circles i and j.

To solve for the coordinates of the center of kth circle if successfully packed, let us consider the following general problem shown in Figure 2.7: Given two fixed circles with radii R_a and R_b and centers at (x_a, y_a) and (x_b, y_b), respectively, find the coordinates of the center of a circle of radius R that is tangent to the two circles.

Assume that $x_a \leq x_b$. To solve this problem, consider the following associated triangle in the xy-plane along with angles as identified in Figure 2.8.

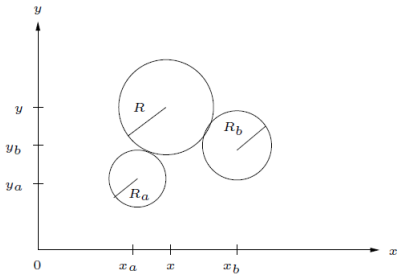


Figure 2.7: Two-Circle Problem

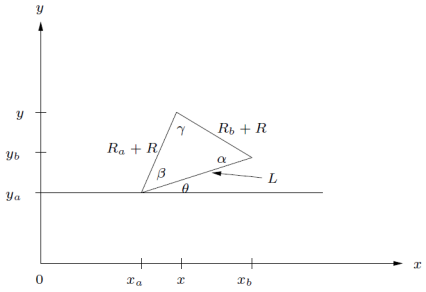


Figure 2.8: Triangle of Two-Circle Problem

L in the figure is the distance between the two fixed centers,

$$L = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}. \tag{2.10}$$

For a solution to exist, it must be satisfied that

$$L < R_a + 2R + R_b. \tag{2.11}$$

By cosine law for triangles,

$$(R_b + R)^2 = (R_a + R)^2 + L^2 - 2(R_a + R)L\cos\beta, \tag{2.12}$$

so that

$$\cos\beta = \frac{L^2 + (R_a + R)^2 - (R_b + R)^2}{2(R_a + R)L}. \tag{2.13}$$

Note also that

$$\cos\theta = \frac{x_b - x_a}{L}, \quad \sin\theta = \frac{y_b - y_a}{L}, \tag{2.14}$$

and

$$\begin{aligned} (R_a + R)\cos(\beta + \theta) &= x - x_a \\ (R_a + R)\sin(\beta + \theta) &= y - y_a, \end{aligned} \tag{2.15}$$

which could be rearranged to solve for x and y,

$$\begin{aligned} x &= x_a + (R_a + R)\cos(\beta + \theta) \\ y &= y_a + (R_a + R)\sin(\beta + \theta). \end{aligned} \tag{2.16}$$

By the Sum Formula for trigonometric functions,

$$\begin{aligned} \cos(\beta + \theta) &= \cos\beta\cos\theta - \sin\beta\sin\theta \\ \sin(\beta + \theta) &= \sin\beta\cos\theta + \sin\theta\cos\beta. \end{aligned} \tag{2.17}$$

All the terms on the right-hand-side of equation 2.16 now can be expressed in terms of the parameters R_a, R_b and L. While converting $\sin\beta$ to $\cos\beta$,

$$\sin\beta = \pm\sqrt{1-\cos^2\beta}, \tag{2.18}$$

the two possible solutions correspond to positions above and below the circles in the diagram. By the GGL algorithm, the solution with negative sign should be discarded because the packing proceeds upwards.

2.1.3 Placing circles by position numbers

In order to pack circles more quantitatively and to keep track of them, position numbers are used to represent the possible positions of a circle to be packed. At this point, it is ignored whether a position is valid for the circle, i.e., at this stage all the constraints imposed by the sides and previously packed circles are not being considered for the enumeration of possible positions.

- **Circle No. 1** The first circle should be placed at Position No. 1, i.e., the lower left corner. Therefore there is only one position available to the first candidate circle.
- **Circle No. 2** Assume Circle No. 1 is already packed. There are five possible positions for placing the second candidate circle.
  1. **Position No. 1**, the lower left corner.
  2. **Position No. 2**, the lower right corner.
  3. **Position No. 3**, tangent to Circle No. 1 and touching Side No. 1.
  4. **Position No. 4**, tangent to Circle No. 1 and touching Side No. 2.
  5. **Position No. 5**, tangent to Circle No. 1 and touching Side No. 3.
- **Circle No. 3** Assume Circles No. 1 and 2 are successfully packed. Obviously the five positions available to Circle No. 2 are also available to Circle No. 3. But now Circle No. 3 can also be packed somewhere tangent to Circle No. 2 and any one of Sides No. 1, 2 and 3, adding three to the number of possible positions for Circle No. 3. Moreover, Circle No. 3 can also be packed tangent to both Circles No. 1 and 2. As such the total number of positions is *nine*.

In general, let f_k denote the number of positions available to circle k, assuming $k-1$ circles have already been packed. Then f_k must satisfy the following recursion relation:

$$f_k = f_{k-1} + 3 + (k-2) = f_{k-1} + k + 1. \tag{2.19}$$

The term f_{k-1} comes from the fact that the positions that were available to Circle No. $k-1$ must be available to Circle No. k. With Circle No. $k-1$ packed, Circle No. k can be packed tangent to Circle No. 2 and any one of Sides No. 1, 2 and 3, and hence 3 appears as the second term. In addition, there are $k-2$ pairings between Circle No. $k-1$ and the $k-2$ previously packed circles, producing the additional $k-2$ positions.

The resulting formula for f_k is

$$f_k = \begin{cases} 1 & k = 1, \\ \frac{1}{2}(k^2 + 3k) & k \geq 2. \end{cases} \tag{2.20}$$

Now suppose that k circles have been packed into the prescribed rectangle and the $(k+1)$st circle is being considered for packing. The first three position numbers available to $(k+1)$st circle are as follows:

- Touching Circle i and Side No. 1: position number $p = 1$.
- Touching Circle i and Side No. 2: position number $p = 2$.
- Touching Circle i and Side No. 3: position number $p = 3$.

More generally, the position numbers available for Circle k with Circles i and j already packed are tabulated in table 2.1. Note that position numbers 1 and 2 respectively represent the lower left corner and lower right corner and s is the variable for side numbers.

According to table 2.1, the position number of the circle placed touching Circle i and Side $s \in \{1, 2, 3\}$ is:

$$\begin{aligned} p &= f_i + s \\ &= \frac{1}{2}(i^2 + 3i) + s. \end{aligned} \tag{2.21}$$

The position number of the circle placed touching Circle i and Circle j, $1 \leq j < i$:

$$\begin{aligned} p &= f_i + 3 + j \\ &= \frac{1}{2}(i^2 + 3i) + j + 3. \end{aligned} \tag{2.22}$$

With the position numbers corresponding to actual positions and coordinates, candidate circles can be packed by assigning a position number to it.

Table 2.1: Position numbers for GGL algorithm for rectangles

i	$s=1$	$s=2$	$s=3$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
1	3	4	5					
2	6	7	8	9				
3	10	11	12	13	14			
4	15	16	17	18	19	20		
5	21	22	23	24	25	26	27	
6	28	29	30	31	32	33	34	35

2.1.4 Position strings

Now suppose that k circles have been packed and the $(k+1)$st circle is being considered for packing. There are

$$f_{k+1} = \frac{1}{2}[(k+1)^2 + 3(k+1)] \tag{2.23}$$

possible positions. One strategy is to try different position numbers starting from $p = 1$ until the $(k+1)$st circle could be packed here satisfying all the constraints. However, it is not desirable to always start at Position No. 1. Therefore, GGL introduces the idea of "position strings"

$$P = (p_1, p_2, \ldots, p_N) \tag{2.24}$$

where N is the number of candidate circles. The element p_k denotes the initial position to be examined when the kth circle is being considered for packing.

Because the first circle is automatically packed at Position No. 1 in this algorithm, we have $p_1 = 1$. An example for position string is

$$p_1 = 1, \quad p_2 = 2, \quad p_3 = 14, \quad p_4 = 10, \quad p_5 = 21, \cdots, p_N = 34. \tag{2.25}$$

In the GGL algorithm, the value of each p_k is randomly assigned or prescribed. For a given $1 < k \leq N$, the number of available positions is f_{n_k}, where n_k is the number of circles that have already been packed in the rectangle when Circle k is being considered. Of course n_k cannot be known ahead of time.

Once a position string P has been defined, it must be decoded in order to determine the initial position of the circle being packed. The following scheme has been used to decode a position number p_k for the placement of Circle No. k.

- If $p_k \leq f_{n_k}$, consider Position p_k for the placement of the kth circle. If $p_k > f_{n_k}$, randomly or schematically assign another value for p_k and proceed.
- If Circle k may be packed at Position p_k, i.e., it satisfies all the constraints, then it is placed there and start considering the next candidate circle, i.e., Circle $k+1$.
- If Circle k cannot be packed at Position p_k, then we examine the next position p_k+1, etc.. If we reach the final possible position f_{n_k} without being able to pack this circle, then we go to Position No. 1 and proceed either until Circle k is packed or, if it cannot be placed in any position then it is concluded that Circle k cannot be packed and we can proceed to pack Circle $k+1$.

To obtain an "optimal" circle-packing in terms of covered area or some other measure, one needs to run this algorithm for a certain number of times and keep track of the best one. If the values of position strings are randomly assigned, the resulted circle-packings would be probabilistically different from each other. Therefore, probabilistically speaking, the more iterations the algorithm is implemented, the better the generated circle-packing would be.

2.1.5 Experimental results

In this subsection we show some experimental results. In Experiments No. 1 and 2, the values of position strings are prescribed while in Experiments No. 3 and 4 the values are randomly assigned at the beginning.

Experiment No. 1 The rectangle has dimensions $A = B = 1$, and there are a total number of $N = 30$ candidate circles with identical radii $R_i = R = 0.1$. We set the position numbers $p(k) = k$. As shown in Figure 2.9, the rectangle is filled up with 25 circles and the packing ratio is 0.7854, which means 78.54% of the rectangle's area is covered by the packed circles.

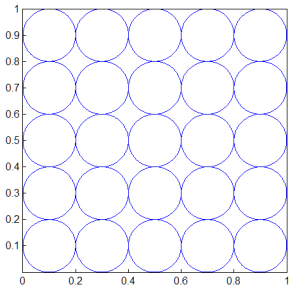


Figure 2.9: GGL Experiment 1 on rectangle

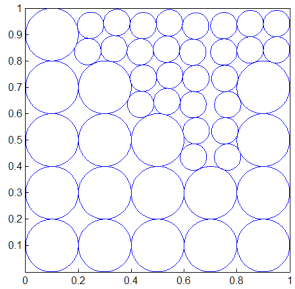


Figure 2.10: GGL Experiment 2 on rectangle

Experiment No. 2 The rectangle still has dimensions $A = B = 1$, and there are a total number of $N = 72$ candidate circles with radii arranged as follows:

$$\begin{aligned} R_1 = R_2 = \cdots = R_{18} = 0.1 \\ R_{19} = R_{20} = \cdots = R_{72} = 0.5R_1 \end{aligned} \tag{2.26}$$

We set the position numbers $p(k) = k$. The result is shown in Figure 2.10, where 46 circles have been packed in the rectangle and the packing ratio is 0.7854.

Experiment No. 3 Same rectangle, and there are a total number of $N = 72$ candidate circles with radii arranged as follows: $(\alpha = 1/\sqrt{3})$

$$\begin{aligned} R_1 = R_2 = \cdots = R_{18} = 0.1 \\ R_{19} = R_{20} = \cdots = R_{72} = \alpha R_1 \end{aligned} \tag{2.27}$$

We assigned a random integer to each $p(k)$ and ran the algorithm for 50 iterations. The best one is shown in Figure 2.11. 35 circles are packed in the rectangle and the packing ratio is 0.7435.

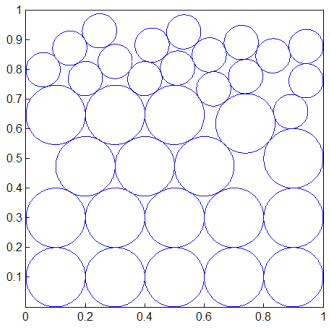


Figure 2.11: GGL Experiment 3 on rectangle

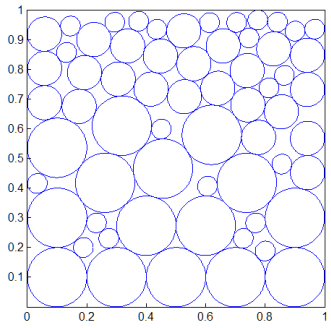


Figure 2.12: GGL Experiment 4 on rectangle

Experiment No. 4 Still the same rectangle, and there are a total number of $N = 105$ candidate circles with radii arranged as follows: $(\alpha = 1/\sqrt{3})$

$$\begin{aligned} R_1 = R_2 = \cdots = R_{15} = 0.1 \\ R_{16} = R_{17} = \cdots = R_{45} = \alpha R_1 \\ R_{46} = R_{47} = \cdots = R_{105} = \alpha^2 R_1 \end{aligned} \tag{2.28}$$

Again, we assigned a random integer to each $p(k)$ and ran the algorithm for 50 iterations. The best one is shown in Figure 2.12. 63 circles are packed in the rectangle and the packing ratio is 0.8063. Among the 50 iterations, the lowest packing ratio is 0.7715.

This is actually random sampling from the solution space and some other strategies, e.g., genetic algorithm, can also be employed. If allowed enough time, as many iterations as possible should be run in order to obtain an optimal circle-packing.

The GGL algorithm of packing circles into a rectangular region provided the method of placing circles by position numbers and this method will be extended for packing circles into arbitrary polygons.

2.2 GGL-based Algorithm of Packing Circles Into an Arbitrary Polygon

In this section, we extend the GGL algorithm of packing circles into a rectangle, using the same scheme and the basic method of position strings employed by the GGL algorithm. Packing different-sized circles into an arbitrary polygon is much more complicated due to the constraints imposed by all the sides and corners.

2.2.1 An arbitrary polygon

Let $D \subset \mathbf{R^2}$ denote the region enclosed by the prescribed polygon with the number of sides denoted by the variable n_s. The coordinates of the vertices are $\mathbf{c_i} = (c_{ix}, c_{iy})$, which are also the coordinates of the corners of the polygon denoted by C_i, $1 \leq i \leq n_s$. Note that the vertices are numbered in the counterclockwise direction and, for convenience, the first vertex of the region D is always at the point $(0, 0)$. Also, a dummy variable $\mathbf{c_{n_s+1}} = (c_{1x}, c_{1y})$ is set for the convenience of representation.

Further, the displacement vectors $\mathbf{v}_i = (v_{ix}, v_{iy})$ represent the movement from the vertex $(0, 0)$ along the boundary ∂D and back to $(0, 0)$ in the counterclockwise direction so that region D is always to the left of the curve. The displacement vectors can be easily calculated using the coordinates of the vertices:

$$\mathbf{v_i} = \mathbf{c}_{i+1} - \mathbf{c}_i, \quad 1 \leq i \leq n_s. \tag{2.29}$$

In component form,

$$\begin{cases} v_{ix} = c_{i+1,x} - c_{ix} \\ v_{iy} = c_{i+1,y} - c_{iy} \end{cases}, \quad 1 \leq i \leq n_s. \tag{2.30}$$

Of course, the net vectorial displacement is $\mathbf{0}$:

$$\sum_{k=1}^{n_s} v_{ix} = 0, \quad \sum_{k=1}^{n_s} v_{iy} = 0. \tag{2.31}$$

Let us look at an example in Figure 2.13.

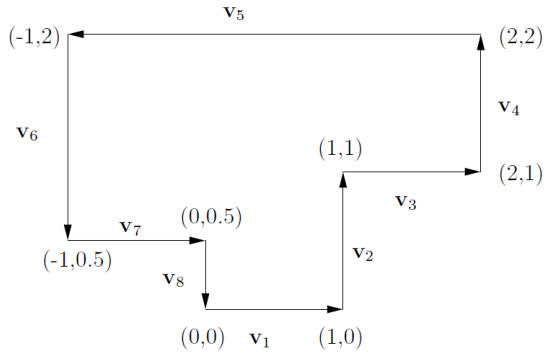


Figure 2.13: Demonstration of vertices and displacement vectors of a polygon

In Figure 2.13 the displacement vectors are as follows:

$$\begin{gathered} \mathbf{v}_1 = (1,0), \quad \mathbf{v}_2 = (0,1), \quad \mathbf{v}_3 = (1,0), \quad \mathbf{v}_4 = (0,1), \\ \mathbf{v}_5 = (-3,0), \quad \mathbf{v}_6 = (0,-1.5), \quad \mathbf{v}_7 = (1,0), \quad \mathbf{v}_8 = (0,-0.5). \end{gathered} \tag{2.32}$$

The equation of each side can also be determined by the coordinates of the vertices:

$$y_i = m_i x + b_i, \quad x \in [c_{ix}, c_{i+1,x}]; \tag{2.33}$$

where m_i and b_i, $1 \leq i \leq n_s$, are respectively the slope and the y-intercept which can be easily computed by $\mathbf{c}_i$. Note that if side i is a vertical line, then the equation should be $x = c_{ix}$.

In order to calculate the packing ratio, one must know the area of the region D first. The area of the region of an arbitrary polygon can be determined by Green's Theorem in the plane.

Assume the region D is simply connected and the boundary ∂D is piecewise smooth, then for a C^1 planar vector field $\mathbf{F}(x,y) = (F_1(x,y), F_2(x,y))$,

$$\oint_C \mathbf{F} \cdot d\mathbf{r} = \iint_D (\frac{\partial F_2}{\partial x} - \frac{\partial F_1}{\partial y}) dA. \tag{2.34}$$

To compute the area of D, to be denoted as $A(D)$, the integrand of the double integral on the right-hand-side must be 1. Here we use the convenient vector field $F = (0, x)$. The line integral on the left may be written as

$$\begin{aligned} \oint_C \mathbf{F} \cdot d\mathbf{r} &= \sum_{i=1}^{n_s} \int_{L_i} \mathbf{F} \cdot d\mathbf{r} \\ &= \sum_{i=1}^{n_s} \int_{L_i} F_1 dx + F_2 dy \\ &= \sum_{i=1}^{n_s} \int_{L_i} x dy. \end{aligned} \tag{2.35}$$

Here the L_i are the piecewise line segments comprising the region D. The net result is the following formula:

$$A(D) = \frac{1}{2}[(c_{2x}+c_{1x})(c_{2y}-c_{1y})+(c_{3x}+c_{2x})(c_{3y}-c_{2y})+\cdots+(c_{1x}+c_{n_s x})(c_{1y}-c_{n_s y})]. \tag{2.36}$$

Returning to the example in Figure 2.13, the area of region D is computed to be

$$\begin{aligned} A(D) &= \frac{1}{2}[(2)(1) + (4)(1) + (-2)(-1.5) + (0)(0.5)] \\ &= \frac{1}{2}(2 + 4 + 3) \\ &= 4.5. \end{aligned} \tag{2.37}$$

2.2.2 Packing corners and position numbers

In Figure 2.13, among all the corners, of course there are some "packing corners" which can hold a circle; and there are some "non-packing corners" which are cusps that cannot hold any circle. As in the GGL algorithm, we shall consider these packing corners as possible positions for placing candidate circles.

Packing corners can be algorithmically distinguished from non-packing corners by means of the displacement vectors. In Figure 2.14 are the packing corners.

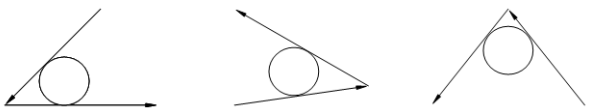


Figure 2.14: Packing corners

Also, in Figure 2.15 are the non-packing corners, indicating the local position of the region D with respect to the line segments.

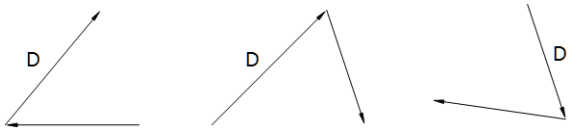


Figure 2.15: Non-packing corners

It can be concluded that the condition for a packing corner $\mathbf{p}_k$ is that

$$\mathbf{v}_{k-1} \times \mathbf{v}_k = A\mathbf{k}, \quad \text{where } A > 0, \tag{2.38}$$

which translates to the condition

$$v_{k-1,x}v_{ky} - v_{k-1,y}v_{kx} > 0. \tag{2.39}$$

Returning to the previous example in Figure 2.13 with $n_s = 8$ sides, it can been seen that there are 6 packing corners, as identified and labeled below:

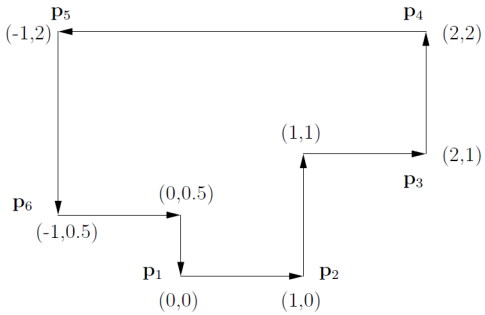


Figure 2.16: An example of packing corners of a polygon

We now have the ingredients of GGL-based algorithm of packing circles into an arbitrary polygon.

1. Packing corners: denoted as $\mathbf{p}_k$, $1 \leq k \leq n_{pc} \leq n_s$. These will define n_{pc} fundamental positions in the algorithm.

2. Sides: denoted as s, $1 \leq s \leq n_s$.

The above ingredients enable us to construct a table of position numbers similar to Table 2.1. Different from GGL algorithm, the first circle can be packed at any packing corner instead of only at the first corner. Suppose that k circles have already been packed into the region, the possible positions to be considered for packing the $(k+1)$st circle are as follows:

1. At the packing corners. Positions at these corners correspond to the position numbers 1 to n_{pc}.
2. Touching Circle i and Side s, $1 \leq s \leq n_s$.
3. Touching two previously packed Circles i and j, $1 \leq j < i$.

Again, let f_k denote the number of positions available to Circle k. As before, f_k must satisfy the following recursion relation,

$$f_k = f_{k-1} + n_s + k - 2. \tag{2.40}$$

The resulting formula is

$$f_k = n_{pc} + n_s(k-1) + \frac{1}{2}(k-1)(k-2). \tag{2.41}$$

Generally, the position numbers available for Circle k with Circles i and j already packed are tabulated in Table 2.2. Note that position numbers from 1 to n_{pc} represent the packing corners and s is the variable for side numbers.

Table 2.2: Position numbers for GGL-based algorithm for polygons

Circle i	$s=1$	...	$s=n_s$	$j=1$	$j=2$	$j=3$
1	$n_{pc}+1$	...	$n_{pc}+n_s$			
2	$n_{pc}+n_s+1$	...	$n_{pc}+2n_s$	$n_{pc}+2n_s+1$		
3	$n_{pc}+2n_s+2$	...	$n_{pc}+3n_s+1$	$n_{pc}+3n_s+2$	$n_{pc}+3n_s+3$	
4	$n_{pc}+3n_s+4$	...	$n_{pc}+4n_s+3$	$n_{pc}+4n_s+4$	$n_{pc}+4n_s+5$	$n_{pc}+4n_s+6$

According to table 2.2, the position number of the circle placed touching Circle i and Side s is:

$$\begin{aligned} p &= f_i + s \\ &= n_{pc} + n_s(i-1) + \frac{1}{2}(i-1)(i-2) + s. \end{aligned} \tag{2.42}$$

The position number of the circle placed touching Circle i and Circle j, $1 \leq j < i$:

$$\begin{aligned} p &= f_i + n_s + j \\ &= n_{pc} + n_s i + \frac{1}{2}(i-1)(i-2) + j. \end{aligned} \tag{2.43}$$

With the position numbers corresponding to actual positions, circles could be packed by assigning a position number to it. For the implementation of this GGL-based algorithm of packing circles into an arbitrary polygon, we again use the position strings and of course the same process described in Section 2.1.4.

2.2.3 Constraints imposed by boundary segments

The circles employed in the packing must satisfy two sets of constraints:

1. They must lie inside the region D, which can be divided into two parts:

   (a) The circle cannot intersect with any side of the polygon at more than one point.

   (b) The center of a circle must lie inside the region D.

   Note that sometimes constraint 1(a) can be satisfied without constraint 1(b) being satisfied. The following figure is an example in which circle c is packed outside the region D but it is tangent to circles a and b and two sides of the polygon.

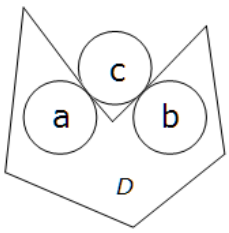


Figure 2.17: A circle lying outside the region satisfying constraint 1(a)

2. The distance between the centers of the two circles must be greater than or equal to the sum of their radii. i.e.,

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq R_i + R_j. \tag{2.44}$$

Constraints 1(a) and 1(b) are nontrivial and need some detailed explanation. Let us consider the segment with endpoints at $C_i = (c_{ix}, c_{iy})$ and $C_{i+1} = (c_{i+1,x}, c_{i+1,y})$ as sketched in Figure 2.18. Now assume that we are considering packing circle C of radius R centered at point P with coordinates (x, y). To examine Constraint 1(a), one has to determine the relative position of the center of circle C with respect to the segment. There are two types of relative positions. The first one is that the center (x, y) lies in the ribbon-like region as shown in Figure 2.18. In this situation, it must be satisfied that the distance between the center of circle C and the segment must be greater than or equal to the radius R.

The distance d between a point (x, y) and a non-vertical line $y = mx + b$ is

$$d = \frac{|y - mx - b|}{\sqrt{1 + m^2}}. \tag{2.45}$$

For the line segment $\overline{C_i C_{i+1}}$,

$$m = \frac{c_{i+1,y} - c_{iy}}{c_{i+1,x} - c_{ix}}, \quad b = c_{iy} - mc_{ix}. \tag{2.46}$$

To check if the point $P = (x, y)$ lie in the ribbon-like region, one only needs to check if the following relations are simultaneously satisfied:

$$\overrightarrow{C_i P} \cdot \overrightarrow{C_i C_{i+1}} \geq 0, \quad \overrightarrow{C_{i+1} P} \cdot \overrightarrow{C_{i+1} C_i} \geq 0. \tag{2.47}$$

Thus checking the first situation of Constraint 1(a) is complete. The second situation is that the center (x, y) does not lie in the ribbon-like region as shown in Figure 2.19.

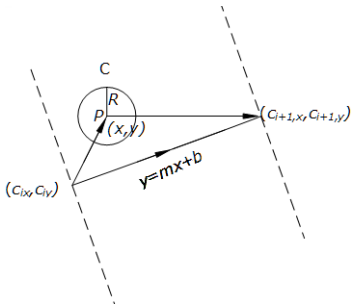


Figure 2.18: One segment and its ribbon-like region

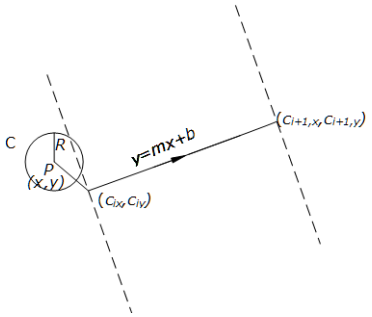


Figure 2.19: The center of circle C outside the ribbon-like region.

In this situation, it is required that the distance between the center $P = (x, y)$ and any one of the endpoints should be no less than the radius R. This condition can be easily

examined and hence checking Constraint 1(a) is complete.

Constraint 1(b) is the famous Point-in-Polygon problem in computational geometry. With Constraint 1(a) satisfied, the center of a circle can only lie either inside or outside the region, indicating that the center cannot be on the boundary. Here we describe an algorithm from the book *Algorithms in combinatorial geometry* [8].

Crossing number algorithm. Given a point P and a polygon, draw a ray starting at P in any direction. If there is an odd number of crossings of the ray with the polygon's edges, then the point P lies inside the polygon; otherwise, the point P lies outside the polygon. An example is shown in Figure 2.20. For simplicity, one can use a vertical or horizontal ray.

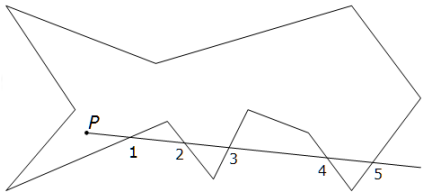


Figure 2.20: A point with odd crossing number

With all the ingredients and methods of examining various constraints, this algorithm can be implemented by a computer program.

2.2.4 Experimental results

Experiment No. 1 The region is the same as in Figure 2.21 and there are a total number of $N = 140$ candidate circles with radii arranged as follows:($\alpha = 1/\sqrt{3}$)

$$\begin{aligned} R_1 = R_2 = \cdots = R_{20} &= 0.2 \\ R_{21} = R_{22} = \cdots = R_{60} &= \alpha R_1 \\ R_{61} = R_{62} = \cdots = R_{140} &= \alpha^2 R_1 \end{aligned} \tag{2.48}$$

The value of each element of the position string is randomly assigned and we ran this algorithm for 50 iterations. The best one is shown in Figure 2.21 where 62 circle have been packed into the region and the packing ratio is 0.8005.

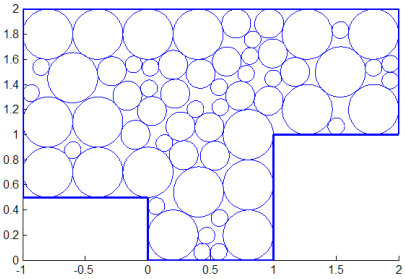


Figure 2.21: GGL-based algorithm Experiment 1

Experiment No. 2 The coordinates of the vertices of the polygon are as follows:

$$\begin{aligned} c_1 = (0,0), \quad c_2 = (1,0), \quad c_3 = (1,1) \\ c_4 = (2,1), \quad c_5 = (2,2), \quad c_6 = (0,2). \end{aligned} \tag{2.49}$$

The total number of $N = 105$ circles with radii arranged as follows: $(\alpha = 1/\sqrt{3})$

$$\begin{aligned} R_1 = R_2 = \cdots = R_{15} = 0.17 \\ R_{16} = R_{17} = \cdots = R_{45} = \alpha R_1 \\ R_{46} = R_{47} = \cdots = R_{105} = \alpha^2 R_1 \end{aligned} \tag{2.50}$$

Again we used randomly assigned position strings and ran the algorithm for 50 iterations. The best one is shown in Figure 2.22 where 58 circle have been packed into the region

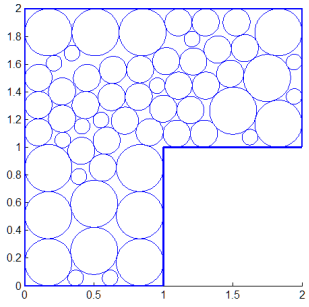


Figure 2.22: GGL-based algorithm Experiment 2

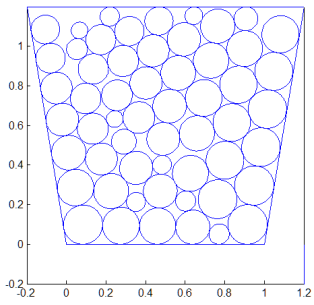


Figure 2.23: GGL-based algorithm Experiment 3

Experiment No. 3 The coordinates of the vertices of the polygon are as follows:

$$c_1 = (0,0), \quad c_2 = (1,0), \quad c_3 = (1.2,1.2), \quad c_4 = (-0.2,1.2) \tag{2.51}$$

The total number of $N = 100$ circles with radii arranged as follows:

$$R_k = 0.1 - 0.0006(k-1), \quad 1 \le k \le N. \tag{2.52}$$

Again we used randomly assigned position strings and ran the algorithm for 50 iterations. The best one is shown in Figure 2.23 where 56 circle have been packed into the region and the packing ratio is 0.8057.

Obviously, in the circle-packings produced by this GGL-based algorithm for arbitrary regions, larger circles are everywhere, i.e., at the corners and the interior. In order for the circle-packings to possess the feature of larger circles being situated primarily in the interior region, we have to explore other strategies of packing circles.

2.3 Reversed-GGL Circle-Packing Algorithm

In the attempt to satisfy the feature of larger circles being situated primarily in the interior region, we now explore a quite different circle-packing algorithm from the GGL-based one, which we call the Reversed-GGL algorithm. Recall that the GGL-based algorithm starts packing circles at packing corners and proceed towards the inside. In Reversed-GGL algorithm, we start packing from the interior of the region and let it proceed towards the boundary. Also, corners and sides are no longer used to assist the packing. Instead, after the first two circles are packed at the interior, the packing proceeds by only "two-circle packing" where a circle must be placed somewhere tangent to two circles.

2.3.1 Scheme and position numbers

Since this algorithm starts packing at the central region of a polygon, it may be desirable to place the first circle at the centroid of the polygon. There are some formulas for calculating the centroid of a polygon in the paper *Calculating the area and centroid of a polygon* [4].

The centroid of a non-self-intersecting closed polygon defined by n vertices $(c_{1x}, c_{1y}), \cdots, (c_{nx}, c_{ny})$ is the point (C_x, C_y), where

$$\begin{aligned} C_x &= \frac{1}{6A}\sum_{i=1}^{n}(c_{ix} + c_{i+1,x})(c_{ix}c_{i+1,y} - c_{i+1,x}c_{i,y}) = 0, \\ C_y &= \frac{1}{6A}\sum_{i=1}^{n}(c_{iy} + c_{i+1,y})(c_{ix}c_{i+1,y} - c_{i+1,x}c_{i,y}) = 0. \end{aligned} \tag{2.53}$$

The formula of computing the area A is given in Equation 2.36 and the vertices are numbered in counterclockwise direction. For convenience of representation, a dummy variable is set to help: $(c_{n+1,x}, c_{n+1,y}) = (c_{1,x}, c_{1,y})$.

The main idea of this algorithm:

1. Start with a set of large circles of radius R. Unless otherwise indicated, pack the first circle at the centroid of the region (C_x, C_y) and the next circle immediately to the right and touching it, i.e., centered at $(C_x + 2R, C_y)$. Then use two-circle packing to pack as many of these large circles as possible into the region.

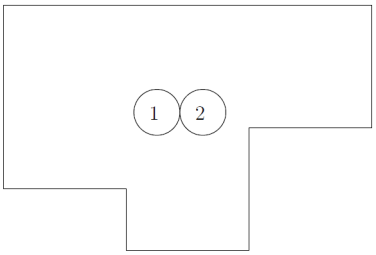


Figure 2.24: Reversed-GGL: First two circles placed near centroid.

2. When Step 1 is completed, pack a set of smaller circles of radius αR, where $0 < \alpha < 1$. Again use two-circle packing to pack as many of these circles as possible into the region.

3. Repeat Step 2 with even smaller circles until no more circles could be packed.

To quantitatively pack the candidate circles, we still use position numbers in this algorithm. Suppose that k circles have already been packed, $k \geq 2$, then there is only one type of positions for the $(k+1)$st circle, i.e., somewhere tangent to two circles. Recall that for the two-circle packing scheme, if a solution exists, then generally two solutions exist. In the GGL-based algorithm, only one solution has been considered, usually the one that packed upwards. However, in the Reversed-GGL algorithm, both solutions are considered for packing and this is reflected in Table 2.3.

It can be seen that there are two position numbers for each j value. An odd number represents a position above two circles while an even number represents the one below two circles. For example, position number 17 represents the position tangent to and above circles 5 and 3.

Table 2.3: Position numbers for Reversed-GGL algorithm

i	$j=1$	$j=1$	$j=2$	$j=2$	$j=3$	$j=3$	$j=4$	$j=4$
2	1	2						
3	3	4	5	6				
4	7	8	9	10	11	12		
5	13	14	15	16	17	18	19	20

It is easy to deduce that the total number of positions available to Circle $k \geq 3$, with circles 1 to $k-1$ having been packed, is

$$f_k = (k-1)(k-2). \tag{2.54}$$

Also, the position number of the circle placed tangent to and above Circles i and j is:

$$\begin{aligned} p &= f_i + 2(j-1) + 1 \\ &= (i-1)(i-2) + 2(j-1) + 1 \\ &= i(i+3) + 2j + 1. \end{aligned} \tag{2.55}$$

Similarly, the position number of the circle placed tangent to and below Circles i and j is:

$$p = i(i+3) + 2j + 2. \tag{2.56}$$

Note that in this algorithm the constraints imposed by boundary segments are exactly the same as described in Section 2.2.3. Now with the steps and the position numbers, we can implement the Reversed-GGL algorithm by means of the position strings and of course the same process described in Section 2.1.4.

2.3.2 Experimental results

In all experiments, the values of position strings are randomly assigned and the algorithm is run 50 iterations.

Experiment No. 1 The coordinates of the vertices of the triangle are as follows:

$$c_1 = (0,0), \quad c_2 = (2,0), \quad c_3 = (0.8,1). \tag{2.57}$$

The total number of $N = 105$ circles with radii arranged as follows: $(\alpha = 1/\sqrt{2})$

$$\begin{aligned} R_1 = R_2 = \cdots = R_{15} = 0.08 \\ R_{16} = R_{17} = \cdots = R_{45} = \alpha R_1 \\ R_{46} = R_{47} = \cdots = R_{105} = \alpha^2 R_1 \end{aligned} \tag{2.58}$$

The best one is shown in Figure 2.25 where 72 circle have been packed into the region and the packing ratio is 0.7389.

Experiment No. 2 The coordinates of the vertices of the polygon are as follows:

$$c_1 = (0, 0), \quad c_2 = (2, 0), \quad c_3 = (1.7, 0.8), \quad c_4 = (0.8, 1). \tag{2.59}$$

The total number of $N = 105$ circles with radii arranged as follows: $(\alpha = 1/\sqrt{2})$

$$\begin{aligned} R_1 = R_2 = \cdots = R_{15} = 0.1 \\ R_{16} = R_{17} = \cdots = R_{45} = \alpha R_1 \\ R_{46} = R_{47} = \cdots = R_{105} = \alpha^2 R_1 \end{aligned} \tag{2.60}$$

The best one is shown in Figure 2.26 where 58 circle have been packed into the region and the packing ratio is 0.7500.

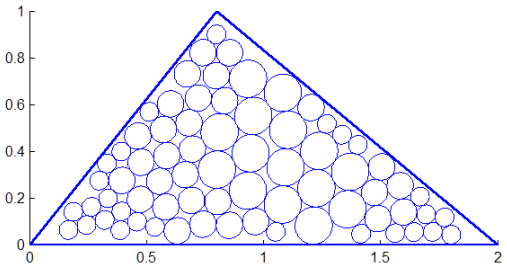


Figure 2.25: Reversed-GGL algorithm Experiment No. 1

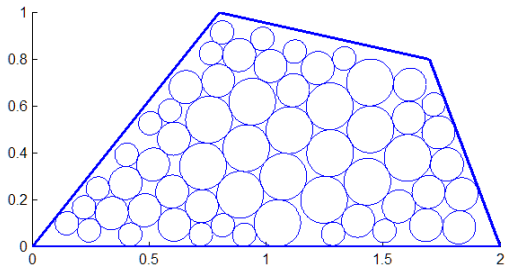


Figure 2.26: Reversed-GGL algorithm Experiment No. 2

Experiment No. 3 The coordinates of the vertices of the polygon are as follows:

$$c_1 = (0, 0), \ c_2 = (1.5, 0), \ c_3 = (1.3, 1.3), \ c_4 = (0.3, 1.8), \ c_5 = (-0.4, 0.9). \tag{2.61}$$

The total number of $N = 105$ circles with radii arranged as follows: $(\alpha = 1/\sqrt{3})$

$$\begin{aligned} R_1 = R_2 = \cdots = R_{15} = 0.15 \\ R_{16} = R_{17} = \cdots = R_{45} = \alpha R_1 \\ R_{46} = R_{47} = \cdots = R_{105} = \alpha^2 R_1 \end{aligned} \tag{2.62}$$

The best one is shown in Figure 2.27 where 68 circle have been packed into the region and the packing ratio is 0.7966.

Experiment No. 4 The coordinates of the vertices of the polygon are as follows:

$$\begin{aligned} c_1 = (0,0), \quad c_2 = (1,0) \quad c_3 = (1.2,1), \quad c_4 = (2,1.4), \\ c_5 = (1,2.5), \quad c_6 = (-1,2), \quad c_7 = (-0.6,0.5). \end{aligned} \tag{2.63}$$

The total number of $N = 140$ circles with radii arranged as follows: $(\alpha = 1/\sqrt{3})$

$$\begin{aligned} R_1 = R_2 = \cdots = R_{20} = 0.2 \\ R_{21} = R_{22} = \cdots = R_{60} = \alpha R_1 \\ R_{61} = R_{62} = \cdots = R_{140} = \alpha^2 R_1 \end{aligned} \tag{2.64}$$

The best one is shown in Figure 2.28 where 72 circle have been packed into the region and the packing ratio is 0.7953.

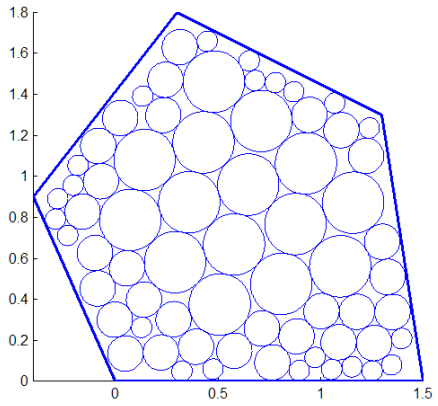


Figure 2.27: Reversed-GGL algorithm Experiment No. 3

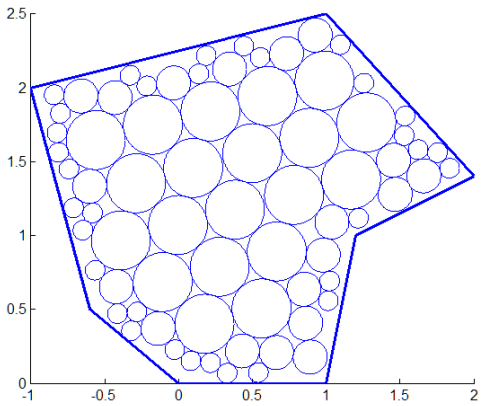


Figure 2.28: Reversed-GGL algorithm Experiment No. 4

Experiment No. 5 In this experiment we approximated an ellipse by a polygon with 32 segments. The radius of the major axis of the ellipse is 5 and the minor radius is 4. The parametric equation can be written as

$$\begin{cases} x = 5 + 5\cos\theta \\ y = 4 + 4\sin\theta \end{cases}, \quad \theta \in [0, 2\pi]. \tag{2.65}$$

Evenly divide the interval $[0, 2\pi]$ into 32 parts:

$$\theta_i = 2\pi i/32, \quad 0 \le i < 32, \tag{2.66}$$

and hence the coordinates of the 32 vertices as follows:

$$\begin{cases} x = 5 + 5\cos\theta_i \\ y = 4 + 4\sin\theta_i \end{cases}, \quad 0 \le i < 32. \tag{2.67}$$

The total number of $N = 140$ circles with radii arranged as follows: $(\alpha = 1/\sqrt{3})$

$$\begin{aligned} R_1 = R_2 = \cdots = R_{20} &= 0.7 \\ R_{21} = R_{22} = \cdots = R_{60} &= \alpha R_1 \\ R_{61} = R_{62} = \cdots = R_{140} &= \alpha^2 R_1 \end{aligned} \tag{2.68}$$

The best one is shown in Figure 2.29 where 80 circle have been packed into the region and the packing ratio is 0.8110. Note that the cluster of the largest circles is mostly at the right part of the ellipse and the upper left part is devoid of the largest circles. The reason is that all the 20 largest circles with radius 0.7 have been used. Apparently, if more large circles were given, the upper left part would be filled with big circles.

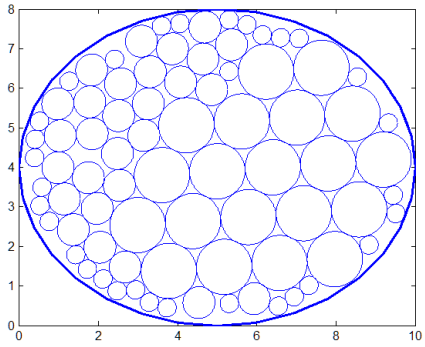


Figure 2.29: Reversed-GGL algorithm Experiment No. 5

Experiment No. 6 In this experiment, the limitation of Reversed-GGL algorithm occurs in this polygon with "needle-like" regions. The coordinates of the vertices of the polygon are as follows:

$$\begin{aligned} c_1 = (0,0), \quad c_2 = (0.5,0) \quad c_3 = (0.5,2), \quad c_4 = (5,2), \\ c_5 = (5,2.5), \quad c_6 = (3.5,2.5), \quad c_7 = (3.5,4), \quad c_8 = (0,4) \end{aligned} \tag{2.69}$$

The total number of $N = 105$ circles with radii arranged as follows: $(\alpha = 1/\sqrt{3})$

$$\begin{aligned} R_1 = R_2 = \cdots = R_{15} &= 0.3 \\ R_{16} = R_{17} = \cdots = R_{45} &= \alpha R_1 \\ R_{46} = R_{47} = \cdots = R_{105} &= \alpha^2 R_1 \end{aligned} \tag{2.70}$$

The best one is shown in Figure 2.30 where 64 circle have been packed into the region and the packing ratio is 0.6750.

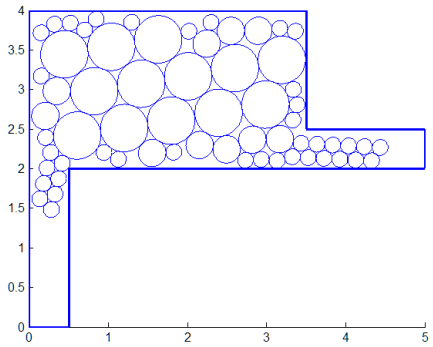


Figure 2.30: Reversed-GGL algorithm Experiment No. 6

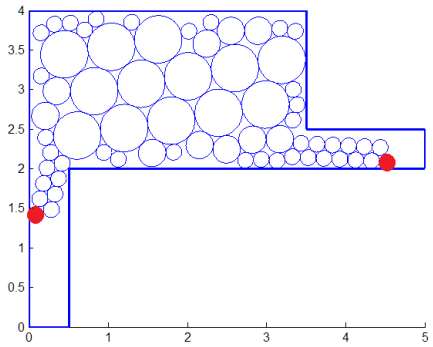


Figure 2.31: Limitation of Reversed-GGL algorithm.

It can be seen in Figure 2.30 that there are some unpacked empty space at the two needle-like regions where obviously more circles with the smallest radius could be placed. This limitation is due to the rule in which only two-circle packing is allowed, which means a circle can only be placed somewhere touching two circles. In Experiment No. 6, the circle-packing in the region could not proceed to reach the needle-like regions because the solid red circles, shown in Figure 2.31, would be partially outside the region if they were packed.

2.4 Reversed-GGL Algorithm with One-Circle Packing

In this section, we add the rule of "one-circle packing" to Reversed-GGL algorithm in order for the packing to reach the needle-like regions. By one-circle packing, a candidate circle is allowed to be placed tangent to only one previously packed circle.

2.4.1 Degree and position numbers

In one-circle packing, a circle can be packed tangent to only one previously packed circle and another variable **degree**, to be denoted by dg, is needed to specify the position of the candidate circle involved in one-circle packing. Draw dg rays starting at the center of a previously packed circle so that the area of this circle is evenly divided into dg parts, then the center of the candidate circle being considered for packing must lie on one of these rays. Note that the first ray is always horizontal extending to the right. An example is shown in Figure 2.32.

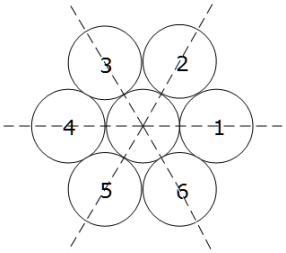


Figure 2.32: Degree of one-circle packing; $dg = 6$.

In general, suppose the center of a previously packed circle of radius R_a is at (x_a, y_a), the candidate circle has radius R and the degree for one-circle packing is dg. If the candidate circle is packed at the qth position around the packed circle, then the coordinates of the center of the candidate circle are:

$$\begin{cases} x = x_a + \cos\beta_q(R_a + R) \\ y = y_a + \sin\beta_q(R_a + R) \end{cases}, \quad 1 \le q \le dg, \tag{2.71}$$

where

$$\beta_q = 2\pi(q-1)/dg. \tag{2.72}$$

With one-circle packing, only the first circle needs to be placed at a prescribed position, e.g., at the centroid. The second circle can be placed by means of one-circle packing and the next candidate circles can be packed by either one-circle packing or two-circle packing, which is different from Reversed-GGL algorithm.

Again, we use position numbers to quantitatively pack the candidate circles. The position numbers available to Circle k with Circle i and Circle j already packed are tabulated in Table 2.4.

Table 2.4: Position numbers for Reversed-GGL algorithm with one-circle packing

i	$q=1$	$\cdots$	$q=dg$	$j=1$	$j=1$	$j=2$	$j=2$
1	1	$\cdots$	dg				
2	$dg+1$	$\cdots$	$2dg$	$2dg$+1	$2dg$+2		
3	$2dg$+3	$\cdots$	$3dg$+2	$3dg$+3	$3dg$+4	$3dg$+5	$3dg$+6

In Table 2.4, the position numbers to the left of the double vertical lines represent positions produced from one-circle packing; otherwise, it is from two-circle packing.

The number of positions available to Circle $k \geq 2$ is

$$f_k = dg + (k-2)(dg + k - 1), \tag{2.73}$$

and the position of a circle placed tangent to circle i and at the qth position around it is

$$p = f_i + q. \tag{2.74}$$

Also, the position number of the circle placed tangent to and above circles i and j is:

$$\begin{aligned} p &= f_i + dg + 2(j-1) + 1 \\ &= 2dg + (i-2)(dg + i - 1) + 2j - 1. \end{aligned} \tag{2.75}$$

Similarly, the position number of the circle placed tangent to and below circles i and j is:

$$p = 2dg + (i-2)(dg + i - 1) + 2j. \tag{2.76}$$

Again, in this algorithm the constraints imposed by boundary segments are exactly the same as in Section 2.2.3. Now we can implement the Reversed-GGL algorithm with one-circle packing by means of the position strings and of course the same process described in Section 2.1.4.

2.4.2 Experimental results

In all experiments, the values of position strings are randomly assigned and the algorithm is run for 100 iterations. Also, the degree for one-circle packing is 6 in all experiments, which corresponds to the case in Figure 2.32

Experiment No. 1 In this experiment we show the advantage of one-circle packing. The polygon and the set of candidate circles are exactly the same as in Experiment No. 6 of Reversed-GGL algorithm in Section 2.3.2. The best one is shown in Figure 2.33 where 71 circles are packed and the packing ratio is 0.7073.

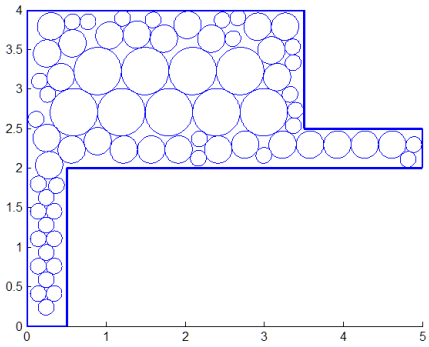


Figure 2.33: Reversed-GGL with one-circle packing Experiment No. 1

As shown in the figure, the needle-like regions are filled with circles due to one-circle packing, which is in contrast with the one in Figure 2.30 generated by Reversed-GGL algorithm with only two-circle packing.

Experiment No. 2 In this experiment we show that this algorithm still cannot restrict big circles in the central region of a polygon. The polygon and the set of candidate circles are exactly the same as in Experiment No. 1 of Reversed-GGL algorithm in Section 2.3.2. The best one is shown in Figure 2.34 where 70 circles are packed and the packing ratio is 0.7289.

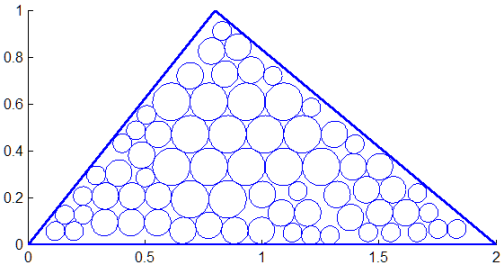


Figure 2.34: Reversed-GGL with one-circle packing Experiment No. 2

As shown in the figure, some of the largest circles are placed very close to the boundary,

which is unsatisfying. However, this problem is easy to fix by adding a set of constraints on the distance between the center of a larger circle and the boundary.

A disadvantage of one-circle packing is that it is less compact than two-circle packing, which means probabilistically much more iterations are needed to generate a circle-packing as good as the ones generated by algorithms employing only two-circle packing.

2.5 Additional constraints

In order to restrict larger circles in the central region of the polygon, the distance from the center of a circle to the boundary of the polygon, to be denoted by d, should be restricted. Suppose a large circle has radius R and a smaller circle has radius αR where $0 < \alpha < 1$. The following constraints are imposed:

$$d \geq R + 2\alpha R \tag{2.77}$$

The desired effect of these constraints is that at least one smaller circle can be packed between the boundary and a larger circle or that there is simply some distance between a larger circle and the boundary. Of course the right-hand-side of the inequality can be adjusted for higher packing efficiency or other purposes and this set of constraints can be applied to any other circle-packing algorithms. Some experiments with these set of constraints are shown below. Note that the degree for one-circle packing is still 6.

Experiment No. 1 Again, the polygon and the set of candidate circles are exactly the same as in Experiment No. 6 of Reversed-GGL algorithm in Section 2.3.2. The constraints in Equation 2.77 are imposed to the circles with radius R_1 and αR_1, where $\alpha = 1/\sqrt{3}$. The best packing after 100 iterations is shown in Figure 2.35 where 83 circles are packed and the packing ratio is 0.7238.

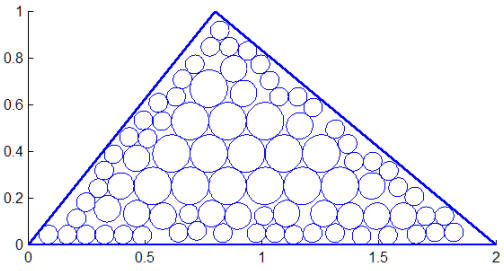


Figure 2.35: Reversed-GGL with one-circle packing and additional constraints Experiment No. 1

As shown in Figure 2.35, the largest circles are all in the central region of the triangle, no medium-sized circle is touching the boundary and the smallest circles are close to or touching the boundary. In contrast, the result in Figure 2.25 does not have this desired feature due to the lack of the additional constraints.

Experiment No. 2 This algorithm is applied to an approximated ellipse in this experiment. The ellipse and its approximation and the set of candidate circles are exactly the same as in Experiment No. 5 of Reversed-GGL algorithm in Section 2.3.2. The constraints in Equation 2.77 are imposed to the circles with radius R_1 and αR_1, where $\alpha = 1/\sqrt{3}$. The best packing after 200 iterations is shown in Figure 2.35 where 109 circles are packed and the packing ratio is 0.7699 which is much lower than 0.8110 in Figure 2.29 produced by Reversed-GGL algorithm with 50 iterations, which is due to the disadvantage of one-circle packing discussed at the end of last section.

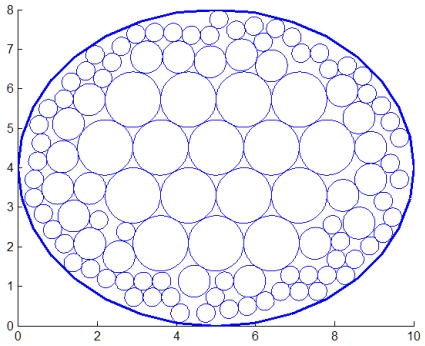


Figure 2.36: Reversed-GGL with one-circle packing and additional constraints Experiment No. 2

As expected, the result in the Figure 2.36 has the desired feature of larger circles being situated in the central part of the region, which is in contrast with the result in Figure 2.29 which is produced by Reversed-GGL algorithm.

Also, recall that there is another desired feature mentioned at the beginning of this chapter on page 6, i.e., there should be as much empty space along the boundary as possible. Up until now, no effort has been made for it, but another novel algorithm called "jiggling", which will be discussed at Section 2.7, is designed to tackle this feature. In addition, the "jiggling" algorithm can significantly accelerate the process of obtaining a decent packing, i.e., many fewer iterations would be needed. Before discussing the powerful "jiggling" algorithm, another method of restricting larger circles in the central region is presented.

2.6 Hybrid Circle-Packing

In this section, we introduce another method of restricting larger circles in the central region of a polygon which can also pack circles at narrow corners which Reversed-GGL cannot. This new algorithm is called Hybrid Circle-packing because it can be divided into two stages where different circle-packing algorithms are employed. In this algorithm, one first packs circles into the region with Reversed-GGL with or without one-circle packing, then removes the outermost layer of circles containing the big circles and then continues packing smallest circles in the candidate set into the region with GGL-based algorithm or Reversed-GGL algorithm with one-circle packing.

2.6.1 Major steps

Step 1. Use Reversed-GGL algorithm to fill the polygon with different-sized circles and obtain the adjacency information about which circles are adjacent to which other circles by the relative positions of the circles with respect to each other.

Step 2. Use a circle with the smallest radius among the set of candidate circles to identify the outermost layer of circles and remove them in the packing. Specifically, tentatively place a smallest circle C_{min} at every possible position tangent to any pair of circles C_a and C_b, if C_{min} intersects with any side of the polygon at more than one point then C_a and C_b are at the outermost layer of the packing. Figure 2.37 provides an illustration.

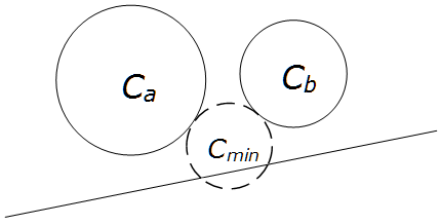


Figure 2.37: Identify the outermost layer of circles

Step 3. Use the adjacency information obtained in Step 1 to identify and count the number of circles at the second outermost layer of circles, i.e., circles touching circles at the first layer and not in the first layer are in the second layer. Two example are shown in Figure 2.38. In the figures, all the circles with an "x" are at the outermost layer, and circles labeled with the number 2 are at the second layer.

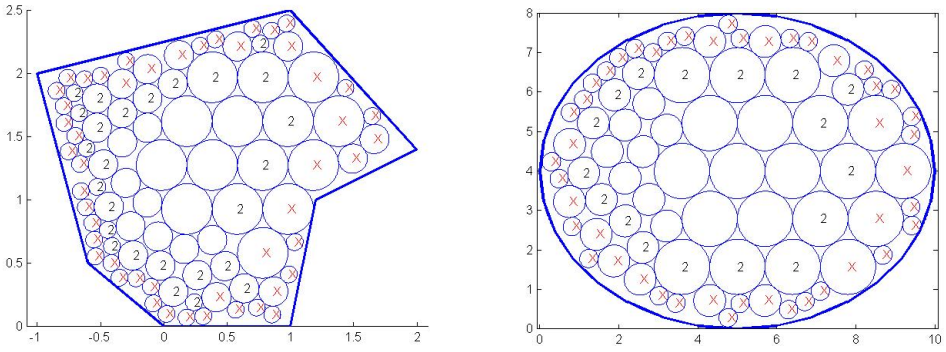


Figure 2.38: Examples of first and second layers of a circle-packing

Step 4. Pack smaller circles between the boundary and the second layer of circles, using GGL-based algorithm for polygons with some modification which will be discussed in the next subsection.

2.6.2 Continue packing with modified GGL-based algorithm

Packing circles into the region between the the boundary and the second layer of circles is more complicated than packing in an "empty" polygon because the boundary of the new region is much more complicated, which is comprised of the boundary of the polygon and the arcs of the previously packed circles.

One way is to "pretend" that no circles have been packed in the polygon and start packing circles and check if they intersect with any previously packed circles. Obviously this scheme would try numerous positions where intersection with previously packed circles occurs, significantly lowering the efficiency of the algorithm and thus wasting a great amount of time and computational resources.

A better way of doing this efficiently is to pack circles based on the circles remaining in the polygon after the outermost layer is removed. This can be done by modifying the position numbers.

Let n_{l2} denote the number of circles at the second outermost layer identified in Step 3. The circles remaining in the polygon which are not at the second outermost layer will not assist in packing in this algorithm. Therefore, there are n_{l2} useful circles in the polygon and the next packed circle will be called the $(n_{l2}+1)$st circle. The position numbers available to the kth circle, $k > n_{l2}$, with useful circles i and j already packed are tabulated in Table 2.5. (Recall that n_s is the number of sides of the polygon and n_{pc} is the number of packing

corners of the polygon.)

Table 2.5: Position numbers for Modified GGL-based algorithm for polygons

i	$s=1$	$\cdots$	$s=n_s$	$j=1$	$\cdots$	$j=n_{l2}-1$	$j=n_{l2}-1$
1	/	$\cdots$	/				
2	/	$\cdots$	/	$n_{pc}+1$			
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$		
n_{l2}	/	$\cdots$	/	M_1+1	$\cdots$	M_2	
n_{l2}+1	M_2+1	$\cdots$	M_2+n_s	M_2+n_s+1	$\cdots$	$\cdots$	$M_2+n_s+n_{l2}$

In the table, $M_1=(n_{l2}-1)(n_{l2}-2)/2+n_{pc}$ and $M_2=M_1+n_{l2}-1$. All the slashes in the table come from the fact that no small or medium-sized circles could simultaneously touch a circle at the second layer and any one of the sides of the polygon.

2.6.3 Demonstration

To demonstrate how this hybrid circle-packing algorithm works, we applied this to the polygon approximating the ellipse which is the same as the one in Figure 2.29. The number of candidate circles is $N=210$ with radii arranged as follows: $(\alpha=1/\sqrt{3})$

$$\begin{aligned} R_1=R_2=\cdots=R_{30}&=0.7\\ R_{31}=R_{32}=\cdots=R_{60}&=\alpha R_1\\ R_{61}=R_{62}=\cdots=R_{120}&=\alpha^2R_1 \end{aligned} \tag{2.78}$$

First, use Reversed-GGL algorithm to pack circles in to the polygon. After only one iteration the result in shown in Figure 2.39 where 57 circles are packed and the packing ratio is 0.8247. In this experiment, the reason why only one iteration is needed here for Reversed-GGL packing is that the remaining circle-packing after removal of the outermost layer has a somewhat fixed pattern where only 19 biggest circles are remaining in the central region.

As shown in the figure, many big circles are very close to or touching the boundary. The outermost and the second outermost layers of circles are identified and also shown in Figure 2.39. As stated, circles with an "x" in it are at the outermost layer and the ones labeled with the number 2 are at the second layer. After removal, there are 19 circles remaining in the polygon as shown in Figure 2.40.

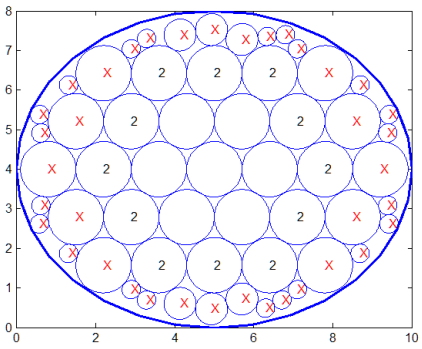


Figure 2.39: Fill the region and identify layers

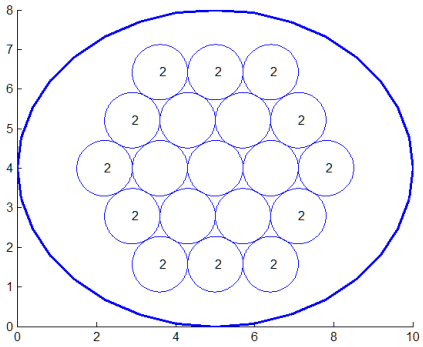


Figure 2.40: After removal of the outermost layer

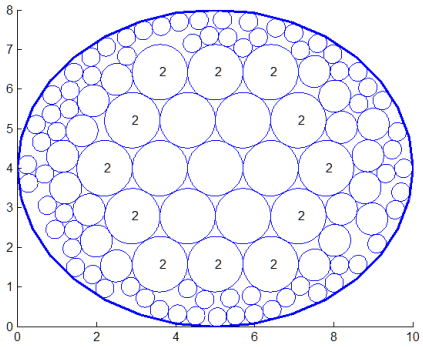


Figure 2.41: Re-packed by modified GGL-based algorithm

Now there is a great amount of empty space released after removal of the first layer of circles, which is to be packed by means of modified GGL-based algorithm described in the last subsection. In the implementation, the following additional constraint is imposed on medium-sized circles of radius $R_m = \alpha R_1$ as in Equation 2.78:

$$d \geq R_m + 1.2\alpha R_m, \tag{2.79}$$

In the constraint, d is the distance between the center of a medium-sized circle and the boundary. Note that the right-hand-side has been adjusted from the one in Equation 2.77.

After 50 iterations of modified GGL-based algorithm, the best one is shown in Figure 2.41 where 108 circles in total have been packed in the polygon and the packing ratio is 0.8110. The packing ratio would be improved if the algorithm is run for more iterations.

This hybrid circle-packing algorithm obviously can restrict larger circles in the central

region but still more empty space need to be released along the boundary, which will be tackled by the next novel algorithm.

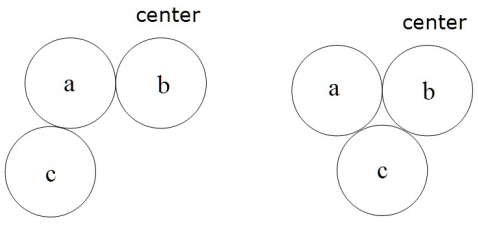
2.7 Jiggling Based on A Given Circle-Packing

In this section we discuss the jiggling algorithm which can release more empty space along the boundary and accelerate the process of obtaining an "optimal" circle-packing. The main idea of this algorithm is to simulate the black hole effect for a given circle-packing, i.e., impose a fictitious force field which attracts all the small and medium-sized circles towards the "center of mass" of a given circle-packing. After the jiggling process there would be more empty space released where more circles could be packed and thus the packing ratio would rise accordingly.

First of all, the formula for computing the center of mass needs to be stated. Suppose N_{pack} is the number of circles packed in the polygon, each of which is centered at (x_i, y_i), where $1 \leq i \leq N_{pack}$. Also, let each circle i packed in the region have mass m_i of the same quantity as its area. Then the center of mass, (x_{cm}, y_{cm}), of a given circle-packing can be computed as follows:

$$x_{cm} = \sum_{i=1}^{N_{pack}} m_i x_i \Big/ \sum_{i=1}^{N_{pack}} m_i, \quad y_{cm} = \sum_{i=1}^{N_{pack}} m_i y_i \Big/ \sum_{i=1}^{N_{pack}} m_i. \tag{2.80}$$

All the small and medium-sized circles will be moved towards the center of mass by a fictitious force imposed by the rule of two-circle packing, which means a circle can be moved only if the new position is closer to the center of mass and if it touches two packed circles. Of course it cannot intersect with any other circles or any side of the polygon. The following figures can illustrate this method.



(a) before (b) after

Figure 2.42: Jiggling

As illustrated in Figure 2.42, circle c was touching only one circle before jiggling. After jiggling, circle c is moved towards the center and now it is touching circles a and b.

Let us apply this algorithm to a real problem. The polygon and the set of candidate circles are exactly the same as in Experiment No.1 of Reversed-GGL algorithm in Section 2.3.2.

Step 1. Use Reversed-GGL algorithm with one-circle packing and additional constraints to fill the polygon with different-sized circles. With one iteration, the result is shown in Figure 2.43 where 78 circles are packed in the triangle and the packing ratio is 0.7188.

Step 2. Calculate the center of mass of the current circle-packing and move medium and small circles towards it. The result is shown in Figure 2.44 where the red dot represent the center of mass. As shown in the figure, some empty space is released and more circles could be packed.

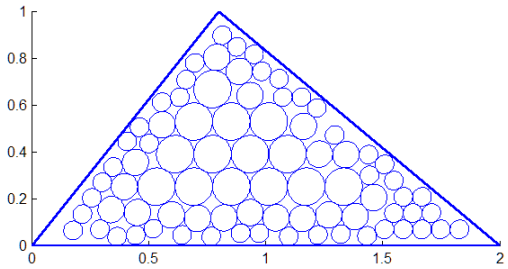


Figure 2.43: Before jiggling

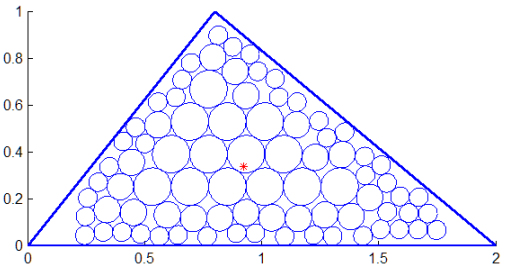


Figure 2.44: After jiggling

Step 3. Use modified GGL-based algorithm to pack the smallest circles of the candidate set into the empty space.

Step 4. Repeat Step 2 and Step 3 until no more circles could be moved.

The final result is shown in Figure 2.45 where 4 more circles are packed and the final packing ratio is 0.7389.

To sum up, in this experiment Reversed-GGL with one-circle packing and additional constraints has been run for only 1 iteration and Step 2 and Step 3 are repeated 3 times.

Obviously the packing ratio 0.7389 is higher than 0.7289 in Experiment No.2 in Section 2.4.2 generated by Reversed-GGL algorithm with one-circle packing with 100 iterations.

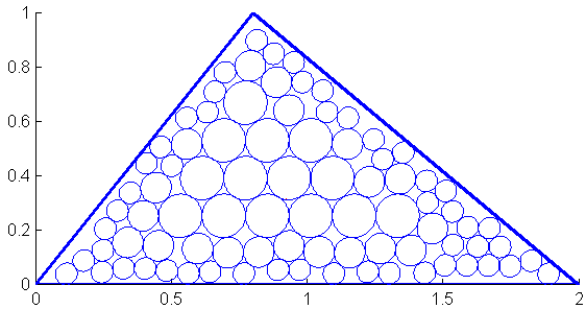


Figure 2.45: Final result of jiggling algorithm

With this jiggling algorithm, the process of obtaining a satisfying circle-packing is significantly accelerated. In addition, since small and medium circles are moved towards the center of mass, more empty space along the boundary is released, which possesses the second feature.

2.8 Summary

In this entire chapter, various novel circle-packing algorithms are described. To generate a circle-packing that possesses the desired features, the following algorithms could be used depending on the prescribed polygon: (additional constraints described in Section 2.5 and the jiggling algorithm must be applied in every algorithm)

1. Reversed-GGL algorithm. (Only for polygons without needle-like regions.)
2. Reversed-GGL algorithm with one-circle packing. (For any polygon.)
3. Hybrid circle-packing algorithm. (Applicable to any polygon if there is no specific requirement on the distance between large circles and the boundary.)

When an algorithm has generated a circle-packing, the jiggling algorithm must be applied to in order to release as much empty space as possible along the boundary.

Chapter 3

Algorithms for Connection

After the tubes have been packed in the blocks obtained by discretizing the prescribed three-dimensional region, they have to be connected to construct a tubular network with one inlet and one outlet at the extreme ends of the region.

In this chapter we consider only the connection problem for tubes with identical radii, and the solution to the problem with different-sized tubes can be extended from this one. From now on, the words "circle" and "tube" are used interchangeably.

We demonstrate in the first section a very simple case where only an endcap connection, composed of two 90-degree bends, is employed. Then we introduce all the operations for connection and the rules of how to apply them in Section 2 where we also illustrate how to construct a network for a fairly complicated case using all the operations. In the next few sections a set of algorithms are developed to generate all the possible solutions and to check feasibility to make sure that there is no dead end or isolated loop.

3.1 A simple tubular network

In this section a very simple case is demonstrated to provide a basic sense of what a tubular network looks like and of how to connect those tubes packed in a prescribed region.

Consider a rectangular cuboid whose front view is as follows where 5 circles are packed:

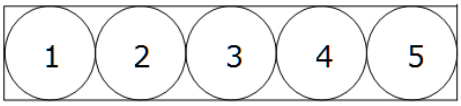
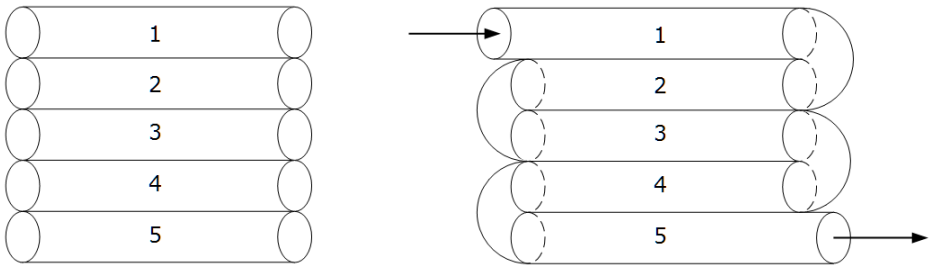


Figure 3.1: Cross section packed with 5 circles

The circles in the rectangle represent tubes extending through the cube, which is shown in Figure 3.2(a). All of them can be connected by only endcap connections except for two tubes that must be left open to serve as the inlet and outlet of this tubular network. An example is shown in Figure 3.2(b).



(a) Before connection (b) After connection

Figure 3.2: 5 tubes in the 3D region

As shown in the figure, at the left end, the pair of tubes 2 and 3 and the pair of tubes 4 and 5 are connected by endcaps; at the right end, the pair of tubes 1 and 2 and the pair of tubes 3 and 4 are connected. Also, tube 1 at the left end is open to serve as the inlet (or outlet) and tube 5 at the right end is the outlet (or inlet).

In this tubular network, the flow of a fluid through the network enters at tube 1 at the left end, travels through all the tubes and exits the network at tube 5 at the right end. Generally, in any valid tubular networks the flow should enter at a certain tube at one end, travel through all the tubes in the network and exit at another tube at another or the same end.

3.2 Operations for connection

In general, there may be various cross sections after the given 3D region has been discretized along the principal direction. In Figure 3.3 is a demonstration where S_i denotes the

interface of two adjacent blocks except for S_1 and S_N which represent two extreme ends. An extreme end or an interface will be uniformly called a *side*, which is different from the "side" of a polygon.

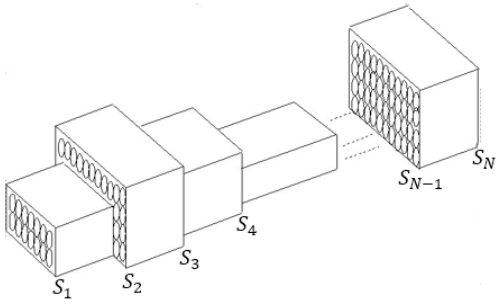


Figure 3.3: Blocks and sides

Suppose all the blocks have already been packed with tubes, i.e., all the cross sections have been packed with circles, then at interface S_i of blocks B_{i-1} and B_i:

- Circles contained in both cross sections of B_{i-1} and B_i are **Common circles**.
- Circles contained only in the cross section of B_{i-1} or B_i are **Boundary circles**.

In this thesis we assume that there is at least one common circle at each interface. An example is shown in Figure 3.4. At this interface of two blocks, the bigger cross section contains 9 circles and the smaller cross section contains only 4 circles. Apparently, circles 1, 2, 4, 5 are common circles as they are in both cross sections; circles 3, 6, 7, 8, 9 are boundary circles as they lie only in the bigger cross section.

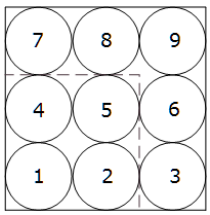


Figure 3.4: Common circles and boundary circles at an interface

For each side, the goal is to connect all the open tubes. Specifically, at an extreme end all the tubes must be connected while at an interface only boundary circles need to be connected because common circles extend through at least two adjacent blocks, leaving no

open tubes at an interface.

A full solution of connecting these tubes in the region is formed by a connection solution at each side. Further, the interfaces are always processed before the two extreme ends, which will be explained in Section 3.3.2.

3.2.1 Elements and connection operations

Three physical elements employed for connection are illustrated in Figure 3.5.

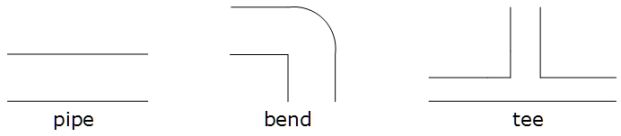


Figure 3.5: Three fundamental physical elements

Three basic operations can be formed by the physical elements:

a). an endcap–two 90-degree bends that connect two adjacent tubes at their ends;

b). a "merge" operation where a tube flows into an adjacent one via a 90-degree bend into a tee; and

c). a shift operation in which a tube is shifted into an adjacent position via a pair of 90-degree bends.

Further, these three basic operations form all the actual operations needed to connect the tubes to construct tubular networks:

1. *Endcap.* This operation connects two adjacent tubes as shown in Figure 3.6. It can be used on open tubes at every side.

2. *Simple merge.* This operation can be used only at an interface and the effect is that two tubes result in one. Figure 3.7 illustrates this operation: tube a merges to tube b. The rule is that tube b must be a common circle and a must be a boundary circle.

3. *Consecutive merges.* This operation can only be used at an interface also and the effect

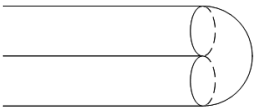


Figure 3.6: An endcap

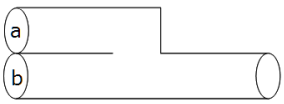


Figure 3.7: A simple merge

is that three tubes end up with one. Figure 3.8 illustrates this operation: tube a merges to tube b and then it merges to tube c. The rule is that tubes a and b must be boundary circles and tube c must be a common circle.

4. *Merge/shift/merge.* Again, this operation can only be used at an interface and the effect is that four tubes end up with two. An example is shown in Figure 3.9 where tube a merges to its adjacent tube b which shifts to the position where tube c used to be, and tube c has to merge to tube d. Note that tubes a and b must be boundary circles and tubes c and d must be common circles.

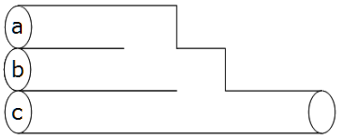


Figure 3.8: Consecutive merges

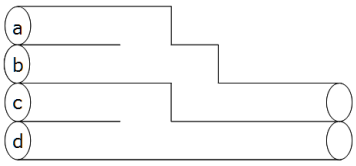
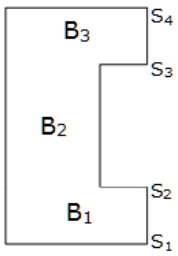


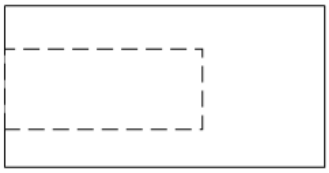
Figure 3.9: merge/shift/merge

3.2.2 A real problem

The top view and front view of the 3D region are as follows:



(a) Top view



(b) Front view

Figure 3.10: The "saddle bag" region

This problem is called the "saddle bag" problem due to the resemblance in shape. This region has 3 blocks, each of which exhibits no variation in cross section. Blocks B_1 and B_3

have the same cross section which is enclosed by solid lines in Figure 3.10(b), and Block B_2 has the smaller cross section which is enclosed by dashed lines. Also, there are two extreme ends S_1 and S_4 and two interfaces S_2 and S_3.

Now we pack circles of identical radii into the two cross sections and the results are shown in Figure 3.11 where 28 circles are packed in the bigger cross section and the small one contains only 8 circles (circles $8 \sim 11$ and $15 \sim 18$). Apparently the 8 circles are common circles and all the other circles are boundary circles.

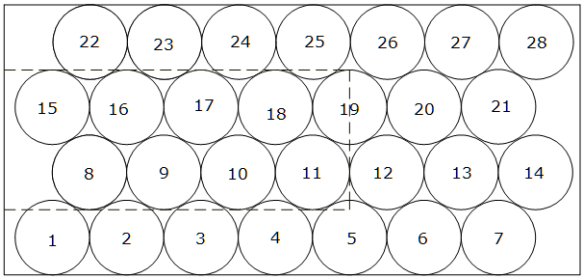


Figure 3.11: Two cross sections packed with circles

A full solution which connects these tubes in the region is formed by a connection solution at each side. Of course there are numerous different solutions and the following is an example. Note that an endcap between tubes i and j is denoted by an ampersand, i.e., $i \,\&\, j$. A merge or shift is denoted by an arrow, e.g., $i \rightarrow j$.

Connection at side S_1.

- Endcaps: 20 & 21, 28 & 27, 26 & 25, 14 & 7, 6 & 5, 24 & 23, 18 & 19, 4 & 3, 10 & 11, 17 & 9, 16 & 22, 15 & 8, 1 & 2.

Connection at side S_2.

- Endcaps: 12 & 20, 21 & 28, 26 & 27, 13 & 14, 7 & 6.
- Simple merges: 22→15, 23→16, 1→8, 19→11.
- Consecutive merges: 3→2→9, 25→24→18.
- Merge/shift/merge: 5→4→10→17.

Connection at side S_3. Same as S_2.

Connection at side S_4. All the endcaps at S_1 plus an endcap 12 & 13.

At side S_1 only 26 tubes are connected and hence the remaining two tubes 12 and 13 are designated as the inlet and outlet of the entire tubular network. Note that in this full solution the inlet and outlet of the entire network are at the same extreme end. However, they can be situated at different extreme ends in some other solutions.

The 2D representation of these four operations are shown in Figure 3.12. The 2D representation of this full solution is illustrated in Figure 3.13 where dashed lines represent endcap connections at extreme ends, solid lines represent different operations at interfaces and the arrows represent the inlet and outlet of the network.

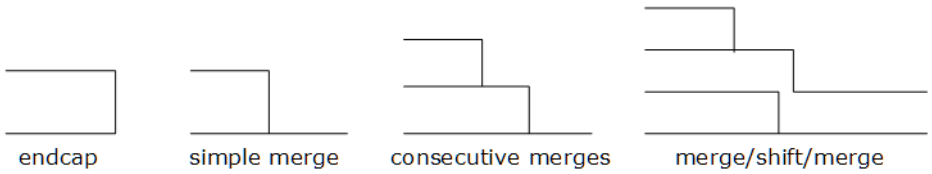


Figure 3.12: 2D representation of all the operations

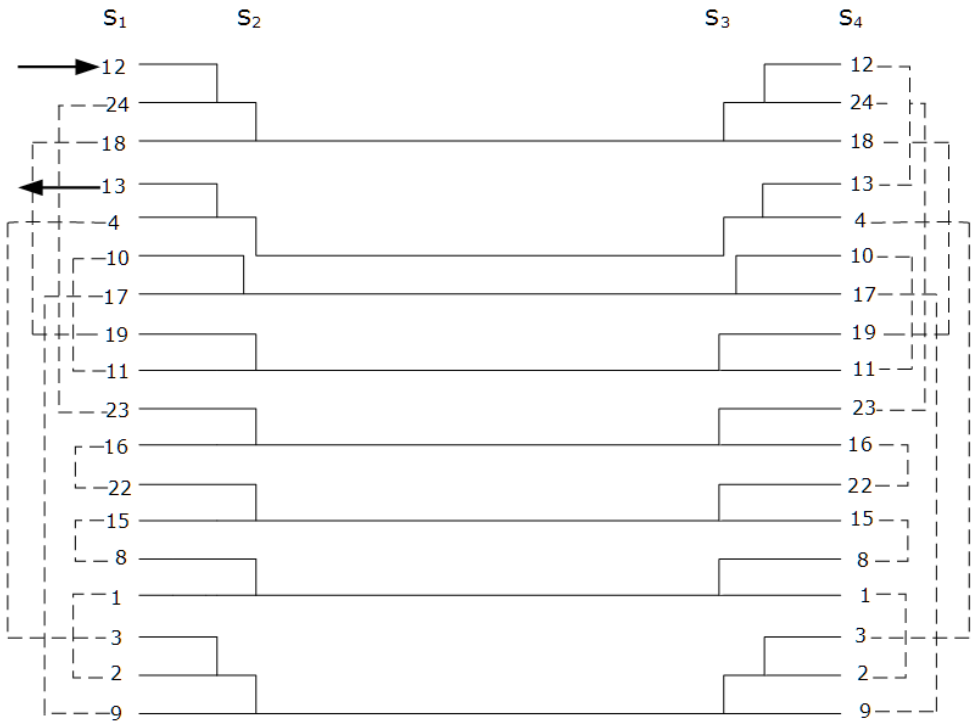


Figure 3.13: 2D representation of a full solution

As shown in the above figure, all the tubes are connected and the flow enters the tubular network at tube 12 at S_1, traverses all the branches and exits at tube 13 also at S_1.

Note that tube 12 at extreme ends S_1 and S_4 represents a long tube composed of 7 tubes connected by endcap connections, and similarly tube 13 represents a long tube composed of 5 tubes.

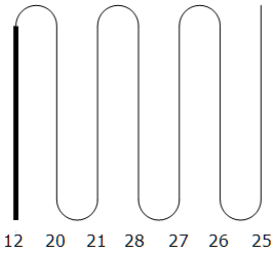


Figure 3.14: Tube 12 composed of 7 tubes

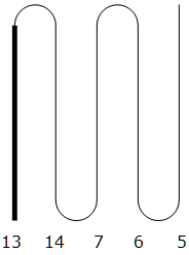


Figure 3.15: Tube 13 composed of 5 tubes

The 3D representation of this full solution is provided in Figure 3.16.

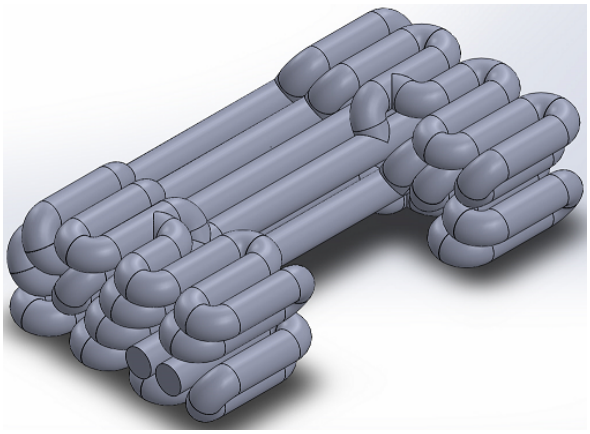


Figure 3.16: 3D representation of a full solution

3.3 Algorithm for extreme ends

As demonstrated in the "saddle bag" problem, a full solution of a connection problem is composed of solutions at each extreme end or interface. In this section, we discuss how to algorithmically generate solutions for each extreme end of a discretized 3D region.

3.3.1 Maximum matching

Before introducing the algorithm, it is important to list a number of definitions, which may be found in Chapter 5 of the book, *Combinatorial Optimization* [6].

Definition 3.1 (Graph)**.** An undirected graph is a triple (V, E, Ψ) where V is the set of vertices (nodes), E is the set of edges and Ψ is a mapping: $E \to \{X \subseteq V : |X| = 2\}$.

Definition 3.2 (Matching)**.** Given a graph G, a matching M in G is a set of pairwise non-adjacent edges; that is, no two edges share a common vertex. A vertex is said to be matched if it is an endpoint of one of the edges in the matching. Otherwise the vertex is unmatched.

Definition 3.3 (Maximum matching)**.** A maximum matching is one that contains the largest possible number of edges.

Definition 3.4 (Maximal matching)**.** A matching M in graph G is maximal if it is not a proper subset of any other matching in G.

Definition 3.5 (Perfect matching)**.** A perfect matching is one that matches all vertices of the graph.

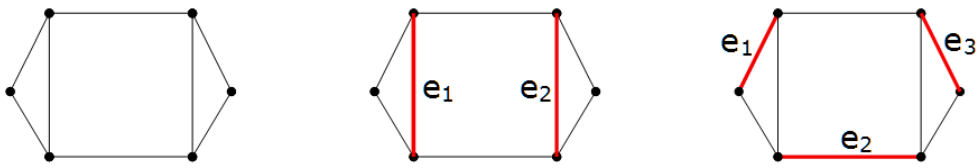


Figure 3.17: Graph G (left) with a maximal matching (middle) and a maximum/perfect matching (right)

Note that every maximum matching is maximal, but not every maximal matching is a maximum matching. In the maximal matching in Figure 3.17, four vertices of the graph are matched by the matching containing 2 edges and no more edge could be added to the matching without sharing a vertex with one of the edges. That matching is only maximal instead of maximum because 2 vertices are still unmatched while they can be covered by the matching at the right part of the figure. As such, the matching at the right is a maximum matching and of course also a perfect matching since all the vertices are matched.

A circle-packing with N circles may be considered to define a graph G with N vertices as follows. Each of the N vertices represents the center of a packed circle. The vertices of circles adjacent to each other are then connected to form edges. We now proceed to

construct endcap connections between tubes. Each endcap connection may be considered to be an edge of a matching M in the graph G. As a result, the problem of connecting tubes at an extreme end may be converted to the problem of finding a maximum matching in the graph G.

Recall that two tubes must be left open at the two extreme ends of the 3D region to serve as the inlet and outlet of the entire tubular network. Further, the inlet and outlet could either lie at the same extreme end out of two or be scattered at two extreme ends.

In the situation where the numbers of tubes at two extreme ends do not possess the same odd-even parity, it is impossible to connect all the tubes with *only endcaps*, except for the inlet and outlet. An example is shown in Figure 3.18.

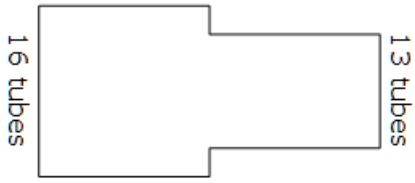


Figure 3.18: Different parity at two extreme ends

In the above figure, there are an even number of tubes at the left end and an odd number of tubes at the right end. No matter how the inlet and outlet are arranged at the two ends, there must be an extreme end where an odd number of tubes need to be connected by endcaps, which is impossible. To resolve this, we can discard this set of circle-packings and produce a new one which possesses the same parity at two extreme ends.

At this point, the reader may notice that an alternative way is to merge two tubes into one at the extreme end with an odd number of tubes that need endcap connections, and then all the open tubes can be connected by only endcaps. However, recall that we mentioned in Section 3.2.1 that a merge operation is allowed only at an interface and thus this alternative way of obtaining same parity is not allowed. In addition, this method, if allowed, could increase the complexity of the entire problem enormously and some space would be lost due to a merge operation immediately before an endcap.

There are two situations of same odd-even parity

1. The numbers of tubes at both ends are *odd*. In this case, we first find a maximum matching and one tube will be left unmatched which will serve as an inlet or outlet.

2. The numbers of tubes at both ends are *even*. In this case, we first designate the inlet and outlet and then find a perfect(maximum) matching in the graph with the inlet and outlet removed. The reason why we do not find a maximum matching first and then choose a pair as the inlet and outlet is that this strategy does not generate solutions in which the inlet and outlet are not adjacent to each other. There are some rules of designating the inlet and outlet which will be discussed in the next subsection.

3.3.2 Minimum degree matching algorithm

We define the *connectivity degree* of each tube to be the number of neighboring tubes that are available for connection, i.e., the number of tubes to which the tube may be connected. The connectivity degree of a tube depends upon the number of adjacent tubes and the connection solution at a neighboring interface.

Suppose A is an extreme end and B is a neighboring interface. Every operation at B can affect the connectivity degree of a tube at A, which explains why interfaces are always processed before extreme ends. The effects of the operations at B are listed below.

1. Endcap connection. A pair of tubes connected by an endcap at interface B cannot be connected by an endcap at the extreme end A. The reason is that they would form an isolated loop from other tubes and the flow cannot reach this part.

2. Simple merge. If at interface B tube a merges into tube b, then they cannot be connected at extreme end A.

3. Consecutive merges. If at interface B tubes a, b and c are in a group of consecutive merges as in Figure 3.8, then tubes a and b cannot be connected at extreme end A.

4. Merge/shift/merge. If at interface B tubes a, b, c and d are in merge/shift/merge as in Figure 3.9, then tubes a and b cannot be connected at extreme end A and also tubes c and d cannot be connected.

The reason for items 2, 3 and 4 is that the forbidden endap connection would result in dead ends, which will be discussed in Section 3.5.2, if they were allowed.

Note that in some cases an operation at an interface could also impose constraints on the connections at an adjacent interface. For example, in the figure below, if tubes a and

b are connected by an endcap at interface S_i, then at interface S_{i+1} tubes a and b cannot be connected by an endcap since the result would be a closed loop that is not connected to the rest of the network.

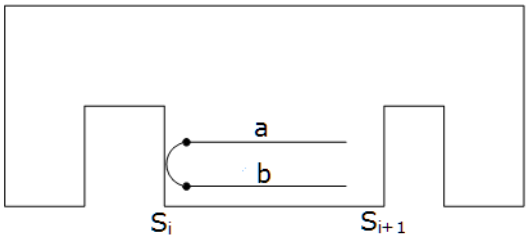


Figure 3.19: Connection at an interface imposing constraints on another one

The following is an example illustrating the connectivity degree of each tube.

The region is discretized into two blocks B_1 and B_2 and it has two extreme ends S_1 and S_3 and one interface S_2. The cross section of the block B_1 is larger than that of block B_2. A novel representation of the connection solution at the only interface S_2 is shown in Figure 3.20.

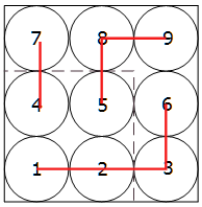


Figure 3.20: Connection solution at the interface S_2

In this method of representation, lines connecting 2 circles represent either an endcap connection or a simple merge, i.e., if one of them is a common circle then the line represents a simple merge, otherwise it is an endcap connection. Red lines connecting 3 circles represent consecutive merges and the ones connecting 4 circles represent merge/shift/shift.

Still, the connection solution in this figure is explained: Simple merge: 7→4; Consecutive merges: 9→8→5; Merge/shift/merge: 6→3→2→1;

Now the connectivity degrees of the 9 tubes at extreme end S_1 can be determined. The connectivity degrees of a few tubes are calculated below as an illustration.

- **Tube 1:** There are two tubes adjacent to Tube 1: Tubes 2 and 4. Since at interface S_2 Tube 2 is merging into Tube 1, these two tubes cannot be connected by an endcap

at extreme end S_1. Tube 4 can be connected to Tube 1 by an endcap and thus the degree of Tube 1 is 1.

- **Tube 2:** There are three tubes adjacent to Tube 2: Tubes 1,3 and 5. As discussed immediately above, Tube 1 cannot be connected to Tube 2. Tube 3 and Tube 5 are available to Tube 2 and thus the connectivity degree of Tube 2 is 2.
- **Tube 3:** Two tubes are adjacent to Tube 3: Tubes 2 and 6. Since Tube 6 is merging into Tube 3, it is not available. Tube 3 was already shown to be available to Tube 2, so the reverse is true. Therefore, the connectivity degree of Tube 3 is 1.
- **Tube 4:** Three tubes are adjacent to Tube 4: Tubes 1,5 and 7. Tube 1 and Tube 5 are available for endcap connection. Tube 7 is merging into Tube 4 so it is unavailable. Therefore, the connectivity degree of Tube 4 is 2.
- **Tube 5:** Situated in the center, this tube has the greatest number of adjacent tubes: Tubes 2, 4, 6 and 8, which are all available. Therefore, the connectivity degree of Tube 5 is 4.

All the connectivity degrees are tabulated in Table 3.2.

Table 3.1: Initial availability list for tubes at side S_1

Tube	Available tubes	Connectivity degree
1	4	1
2	3, 5	2
3	2	1
4	1, 5	2
5	2, 4, 6, 8	4
6	5, 9	2
7	8	1
8	5, 7	2
9	6	1

Now with the connectivity degrees the *Minimum Degree Matching Algorithm* can be described below. Note that the steps are for finding only one matching.

Step 1. If the numbers of tubes at two extreme ends are both even, choose the inlet and outlet and mark them as unavailable to any other tubes at the same end. If odd, proceed to the next step.

Step 2. Determine the available tubes and therefore the initial connectivity degree for each tube.

Step 3. Randomly(or systematically) choose one tube with minimum connectivity degree and one of its available tubes for connection, add this pair to the matching.

Step 4. Delete the connected tubes from the availability list and update the connectivity degrees of relevant tubes.

Step 5. Repeat step 3 and step 4, i.e., choosing pairs and updating the list, until no more tubes can be connected.

Returning to the example in Figure 3.20, the numbers of the tubes at both ends are odd and hence Step 1 can be skipped. Further, the initial availability list has been made in Table 3.2 and thus Step 2 is completed.

In Step 3, the tubes with non-zero minimum degree are 1, 3, 7, 9. Randomly choose one, for instance, 3. Its only available tube 2 should be connected to it and now we have a pair {3,2}.

Since tubes 2 and 3 are already connected to each other, they should not be available to any other tubes and the availability list is updated below, which completes Step 4.

Table 3.2: Availability list for tubes at side S_1 after {3,2}

Tube	Available tubes	Connectivity degree
1	4	1
4	1, 5	2
5	4, 6, 8	3
6	5, 9	2
7	8	1
8	5, 7	2
9	6	1

Now repeat Step 3 and Step 4 until a connection solution is obtained, for instance: $M = \{\{3,2\}, \{1,4\}, \{7,8\}, \{5,6\}\}$. In this solution, Tube 9 is unmatched so it can be an inlet or outlet of the entire tubular network.

The above is an example where the numbers of tubes at two extreme ends are both odd

and the inlet and outlet do not need to be actively designated. Recall that in the case where the numbers are both even, the inlet and outlet do need to be designated in Step 1 and there are some rules for doing this:

1. Choosing inlet/outlet should not exhaust the available tubes of any other tube. For instance, in the example above, tubes 3 and 5 are not allowed to be chosen as inlet and outlet because this would isolate tube 2.

2. Choosing inlet/outlet should not lead to the situation where two different tubes have the same unique available tube for connection. For instance, in the same example above, if 1 and 3 are chosen as inlet and outlet, tube 2 and 4 would both have tube 5 as the only available tube, which is not allowed.

Note that this algorithm only gives one matching which is not guaranteed to be a maximum matching. However, our goal is not to find a maximum matching for one pass. Instead, the goal is to enumerate all the maximum matchings of a given graph. Fortunately, this *Minimum Degree Matching Algorithm* combined with *Depth-first Search* is able to enumerate all the maximum matchings in a graph.

According to the paper [3] by Bert Besser, this algorithm can be implemented in linear time $O(|V|+|E|)$ where $|V|$ represents the number of vertices and $|E|$ the number of edges. Also, in experiments of Frieze et al.[11] on random cubic graphs, in which a perfect matching is guaranteed to exist, the *Minimum Degree Matching Algorithm* left only 10 out of 10^6 vertices unmatched. On random graphs of small constant average degree Magun[21] observed that this algorithm produces extremely few lost edges in comparison with an optimal solution.

Even though the algorithm can sometimes produce a maximal matching instead of a maximum one, it is very easy to eliminate those maximal matchings by counting the number of edges in the resulted matching.

To sum up, the *Minimum Degree Matching Algorithm* is not guaranteed to produce a maximum matching of a given graph, but it can efficiently enumerate all the maximum matchings when combined with *Depth-first Search*.

3.4 Algorithm for interfaces

The problem of connecting tubes at an interface is much more complicated than connecting

them at extreme ends. In this section, we describe the *Unified Minimum Degree Matching Algorithm*, so named because all the four operations introduced in Section 3.2.1 will be dealt in the same way. Recall that at an interface only boundary circles need to be connected because common circles extend through at least two adjacent blocks, leaving no open tubes at an interface.

The idea of the algorithm of generating connection solutions at an interface is similar to that of the *Minimum degree matching algorithm*, i.e., repeatedly picking edges incident to nodes of current minimum non-zero connectivity degree. Further, all the connections are made in pairs and will be decoded to obtain a meaningful solution.

The steps of this algorithm are listed below and will be explained later.

Step 1. Identify the layer-rank of each tube at the interface being processed.

Step 2. Make the initial availability list, i.e., calculate connectivity degrees and connectivity values.

Step 3. Randomly(or systematically) choose one tube with the highest priority, i.e., minimum degree or some other measure which will be discussed later. Add this tube and one of its available tubes as a pair to the matching.

Step 4. Update the availability list based on a set of rules.

Step 5. Repeat step 3 and step 4, i.e., choosing pairs and updating the list. If all the boundary circles are connected, one can either stop and get a solution or continue to obtain another solution where more common circles are involved. The process must stop when no more circles can be connected.

Step 6. Decode the raw solution and obtain a meaningful one.

3.4.1 Layer-rank and availability list

Recall that at interface S_i of blocks B_{i-1} and B_i, circles contained in both cross sections of the two blocks are common circles and the ones contained only in one cross section are boundary circles. Based on these concepts, the layer-rank of each circle, to be denoted by L_r, is defined below:

- First-layer boundary circle, $L_r = 1$: Boundary circles touching common circles. For convenience of representation it will be called $1B$ circle.
- Other boundary circle, $L_r = 0$: Boundary circles not touching common circles. It will be called $2B$ circle.
- First-layer common circle, $L_r = 2$: Common circles touching boundary circles. It will be called $1C$ circle.
- Second-layer common circle, $L_r = 3$: Common circles touching $1C$ circles. It will be called 2C circle.
- Other common circles, $L_r = +\infty$, which are not involved in this algorithm because it is unnecessary and impossible for it to be connected at an interface.

We now introduce a new concept into the availability list for this algorithm – the **Connectivity Value**, a measure of the availability of a tube to other tubes. A $1B$ or $1C$ circle has an initial connectivity value of 1. A first-layer circle could be at the middle of a group of consecutive merges or merge/shift/merge, i.e., it could be connected to two tubes. For instance, in Figure 3.8 tube b must be 1B and it can be connected to both a and c in this form: $\{\{a, b\}, \{b, c\}\}$.

A $2B$ or $2C$ circle has an initial connectivity value of 0.5. This is because a second-layer boundary or common circle can be connected to only one circle.

The **connectivity degree** in this algorithm is calculated differently. Only boundary circles are assigned a meaningful connectivity degree which is the number of adjacent circles. A common circle does not have a meaningful connectivity degree because it does not have to be connected at an interface. Instead, its connectivity value will be assigned a large enough number, e.g., 1000, which means it never has the priority of getting connected before all the boundary circles are connected.

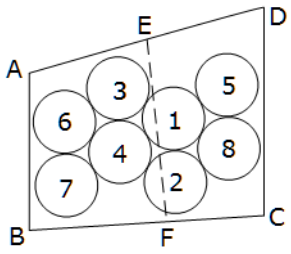


Figure 3.21: An example for solutions at an interface

Now the list of elements of an availability list in this algorithm is complete and an illustrative example is provided in Figure 3.21 which is the interface of two blocks. The larger block has the cross section $''ABCD''$ which contains 8 circles and the smaller one has the cross section $''ABFE''$ which contains only 4 circles: 3, 4, 6, 7.

The initial availability list for the tubes at this interface is made in Table 3.3.

Table 3.3: Initial availability list for tubes at the interface

Tube	Layer-rank	Connectivity value	Available tubes	Connectivity degree
1	1 ($1B$)	1	2, 3, 4, 5, 8	5
2	1 ($1B$)	1	1, 4, 8	3
3	2 ($1C$)	1	1, 4, 6	1000
4	2 ($1C$)	1	1, 2, 3, 6, 7	1000
5	0 ($2B$)	0.5	1, 8	2
6	3 ($2C$)	0.5	3, 4, 7	1000
7	3 ($2C$)	0.5	4, 6	1000
8	0 ($2B$)	0.5	1, 2, 5	3

3.4.2 Rules for connection and updating availability list

There are a set of rules for making connections in this algorithm.

1. When there exists at least one boundary circle that is completely unconnected, the circles with the highest priority of getting connected are the ones with the current minimum degree.

2. If each boundary circle has been connected to at least one circle, then the circles with the highest priority are the ones that are already connected.

3. A $1B$ circle has three forms of connection.

(a) Connected to only one boundary circle, which forms an endcap connection.

(b) Connected to only one common circle, which forms a simple merge.

(c) Simultaneously connected to one boundary circle C_B and one common circle, which forms a part of a group of consecutive merges or merge/shift/merge, on the condition that C_B is not connected to any other circle.

4. A $2B$ circle can be connected to only one boundary circle. The reason is that it is two layers away from common circles and in a group of consecutive merges or merge/shift/merge there should be at most 2 boundary circles.

5. A $1C$ circle has three possible situations.

(a) Not connected at all. (A common circle does not have to get connected.)

(b) Connected to only one boundary circle, which forms a simple merge.

(c) Simultaneously connected to one boundary circle C_B and one common circle C_C, which gives a part of a group of merge/shift/merge. The condition is that C_B is already or will be connected to another boundary circle.

6. A $2C$ circle has two possible situations.

(a) Not connected at all.

(b) Connected to only one common circle C_C only when C_C is the common circle of a group of consecutive merges. The resulted operation is a merge/shift/merge.

Based on the rules of making connections, the rules for updating the availability list are listed below:

7. Deduct 0.5 from the connectivity value if a tube is connected. If the connectivity value becomes zero then delete this tube from the list, i.e., mark this tube as unavailable to any other tubes and deduct 1 from the degree of relevant boundary circles that are not connected.

8. If a tube is connected, set its degree to 1000 so that it does not have the priority before all the boundary circles have been connected.

9. Updating the availability list should conform to the rules of making connections.

Now let us apply these rules to the example in Figure 3.21 with the initial availability list displayed in Table 3.3.

Pair 1. In the initial availability list, tube 5 has the minimum degree 2. Randomly choose a tube between tubes 1 and 8 we have the first pair: {5,1}.

Set the degrees of tubes 1 and 5 to 1000. Deduct 0.5 from the connectivity value of each tube and now tube 5 has zero connectivity value. Therefore tube 5 should be deleted from the list.

According to rule 3(a), tube 1 can no more be connected to any other boundary circles. Therefore, tubes 2 and 8 should be deleted from its available tubes and vice versa. The updated availability list is shown below.

Table 3.4: Availability list for tubes at the interface after {5,1}

Tube	Layer-rank	Connectivity value	Available tubes	Connectivity degree
1	1 ($1B$)	0.5	3, 4	1000
2	1 ($1B$)	1	4, 8	2
3	2 ($1C$)	1	1, 4, 6	1000
4	2 ($1C$)	1	1, 2, 3, 6, 7	1000
6	3 ($2C$)	0.5	3, 4, 7	1000
7	3 ($2C$)	0.5	4, 6	1000
8	0 ($2B$)	0.5	2	1

Pair 2. Now tube 8 has the minimum degree 1 so the second pair is {8,2}.

Set the degrees of tubes 8 and 2 to 1000. Decrease connectivity values and tube 8 should be deleted from the list. The updated availability list is shown below.

Table 3.5: Availability list for tubes at the interface after {8,2}

Tube	Layer-rank	Connectivity value	Available tubes	Connectivity degree
1	1 ($1B$)	0.5	3, 4	1000
2	1 ($1B$)	0.5	4	1000
3	2 ($1C$)	1	1, 4, 6	1000
4	2 ($1C$)	1	1, 2, 3, 6, 7	1000
6	3 ($2C$)	0.5	3, 4, 7	1000
7	3 ($2C$)	0.5	4, 6	1000

Note that now all the boundary circles have been connected and there are two pairs in the solution: $M = \{\{5, 1\}, \{8, 2\}\}$. This is a solution at this interface, i.e., two endcap connections. One can either stop here and accept this solution or continue to obtain a longer solution involving more tubes as demonstrated below.

Pair 3. As all the boundary circles have been connected by now, all the tubes that are connected have the same priority of getting connected to one more tube.

Currently in the list, tubes 1 and 2 are connected. Randomly choose a tube between 1 and 2 and one of its available tubes we have the next pair: {1,3}.

Decrease connectivity values and tube 1 should be deleted from the list. The updated availability list is shown below.

Table 3.6: Availability list for tubes at the interface after {1,3}

Tube	Layer-rank	Connectivity value	Available tubes	Connectivity degree
2	1 ($1B$)	0.5	4	1000
3	2 ($1C$)	0.5	4, 6	1000
4	2 ($1C$)	1	2, 3, 6, 7	1000
6	3 ($2C$)	0.5	3, 4, 7	1000
7	3 ($2C$)	0.5	4, 6	1000

Now the solution is $M = \{\{5,1\},\{8,2\},\{1,3\}\}$ which is a raw solution that needs to be decoded. The steps of decoding a raw solution are described below.

Step 1. Put a pair $\{L, R\}$ in a group. For instance, {5,1} is now a group.

Step 2. Scan through the next pairs $\{L_i, R_i\}$.

If one of L_i and R_i equals L, add the other one to the left of L in the group.

If one of L_i and R_i equals R, add the other one to the right of R in the group.

Otherwise, put $\{L_i, R_i\}$ in a new group and repeat this step until all the pairs are processed.

For example, the pair {8,2} will be a new group and the pair {1,3} will be integrated in the group {5,1} and the resulted group is {5,1,3}.

Step 3. Interpret each group and obtain meaningful connection solutions.

a) A group $\{a, b\}$ represents a simple merge if there is exactly one boundary circle and one common circle. For instance, if a is a common circle then this group represents a simple merge: $b \rightarrow a$.

b) A group $\{a, b\}$ represents an endcap connection if a and b are both boundary circles. Note that it is impossible that a and b are both common circles.

c) A group $\{a, b, c\}$ represents a group of consecutive merges. If a is a common circle the connection is $c \rightarrow b \rightarrow a$; Otherwise the connection is $a \rightarrow b \rightarrow c$.

d) A group $\{a, b, c, d\}$ represents a group of merge/shift/merge. If a is a common circle the connection is $d \rightarrow c \rightarrow b \rightarrow a$; Otherwise the connection is $a \rightarrow b \rightarrow c \rightarrow d$.

In the example, there are two groups: {5,1,3} and {8,2} which respectively represent consecutive merges 5→1→3 and an endcap 8&2.

Currently there are three pairs in the solution, i.e., $M = \{\{5,1\},\{8,2\},\{1,3\}\}$, and one can either stop here and accept this solution or continue to obtain a longer one.

Pair 4. Currently in the list, tubes 2 and 3 are connected. Randomly choose a tube between 2 and 3 and one of its available tubes we have the next pair: {2,4}.

Decrease connectivity values and tube 2 should be deleted from the list.

Also, since tube 4 is now connected to a boundary circle, according to rule 5(b) it cannot be connected to any other boundary circles. Therefore, tubes 3 and 4 are not available to each other any more. The updated availability list is shown below.

Table 3.7: Availability list for tubes at the interface after {2,4}

Tube	Layer-rank	Connectivity value	Available tubes	Connectivity degree
3	2 ($1C$)	0.5	6	1000
4	2 ($1C$)	0.5	6, 7	1000
6	3 ($2C$)	0.5	3, 4, 7	1000
7	3 ($2C$)	0.5	4, 6	1000

Now there are two groups: {5,1,3} and {8,2,4} which respectively represent consecutive merges 5→1→3 and consecutive merges 8→2→4.

Again this process can stop or continue making connections according to those rules and the demonstration stops here.

3.5 Verifcation of feasibility

Connection solutions at each side together form a full solution whose feasibility needs to be verified. Some knowledge in graph theory is applied to complete this. There are three steps of checking feasibility of a potential solution.

Step 1. Construct a graph of the potential solution.

Step 2. Check if this graph is connected, i.e., there should be no isolated loop.

Step 3. Check if there is a bridge in the graph, i.e., there should be no dead end.

If the graph is disconnected or there is a bridge in it, then the solution is infeasible and should be discarded.

3.5.1 Constructing the graph

Given a full solution, the first thing to do is construct a graph according to the connections on each side.

Let us first look at a graph in a sample problem. The top view and front view of the 3D region are shown below.

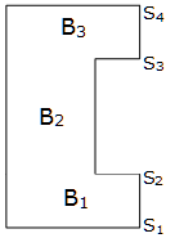


Figure 3.22: Top view of the region–small "saddle bag"

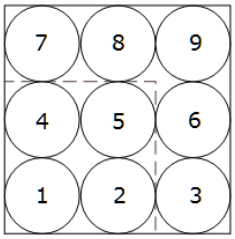


Figure 3.23: Front view and cross sections of the small "saddle bag"

Blocks B_1 and B_3 have the same cross section which contains 9 circles and Block B_2 has the smaller cross section which contains only four circles: 1, 2, 4, 5.

We call this problem "9-4-9" problem which will be discussed again later. One of many solutions is provided here:

Connection at side S_1. Endcaps: 1 & 4, 3 & 2, 7 & 8, 5 & 6.

Connection at side S_2.

- Simple merges: 7→4.
- Consecutive merges: 9→8→5.
- Merge/shift/merge: 6→3→2→1.

Connection at side S_3.

- Endcaps: 7 & 8.

- Simple merges: 3→2.
- Merge/shift/merge: 9→6→5→4.

Connection at side S_4. Endcaps: 3 & 6, 7 & 4, 1 & 2, 9 & 8.

The 2D representation and the graph of this solution are shown in Figure 3.24. There are 28 nodes in total and all the lines between nodes, in whatever form, represent edges of the graph.

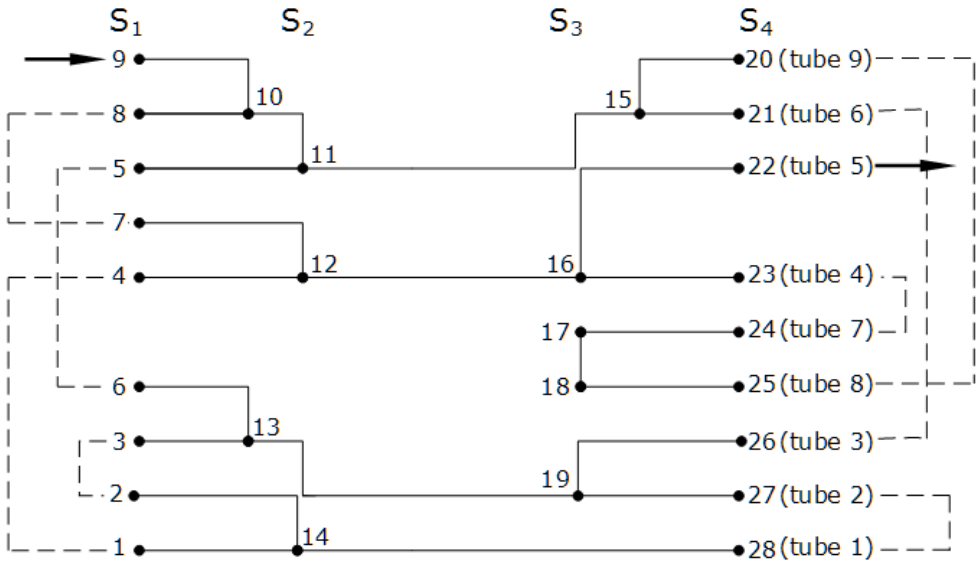


Figure 3.24: 2D representation and the graph of a sample solution for the 9-4-9 problem

In general cases, assume the discretized region has S sides, each of which contains N_i circles, $1 \leq i \leq S$. The following is the method of constructing the graph of a solution.

- Add and number nodes for tubes at each side, which gives $(N_1 + N_2 + \cdots + N_S)$ nodes.
- Add an edge for each tube, i.e., add the edge connecting two endpoints of a tube. An example is the edge {17,24} in the figure above.
- Add an edge for an endcap connection {a,b} at any side S_i: Find the corresponding node numbers of tubes a and b at side S_i, to be denoted by a_{num} and b_{num}, and add the edge $\{a_{num}, b_{num}\}$.

  For example, in the graph shown above, at side S_3 tubes 7 and 8 are connected by

an endcap and their corresponding node numbers at side S_3 are respectively 17 and 18. Therefore, the edge {17,18} is added.

- For a simple merge $a \rightarrow b$, set node a_{num} to the junction point of the merge operation, which is shown in Figure 3.25.

  Then break the edge $\{x_2, x_3\}$ into two edges, i.e., delete $\{x_2, x_3\}$ and add $\{x_2, a_{num}\}$ and $\{a_{num}, x_3\}$.

  Note that x_1, x_2 and x_3 are the node numbers of tubes a and b at adjacent sides and they can be found by connection solutions and previously added nodes and edges.

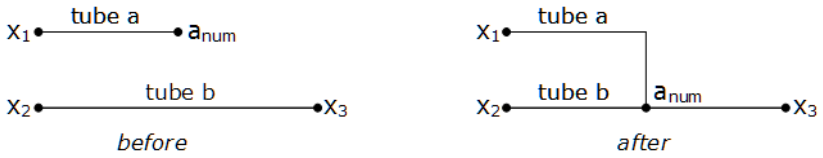


Figure 3.25: Nodes and edges for a simple merge

- For consecutive merges $a \rightarrow b \rightarrow c$, set nodes a_{num} and b_{num} to the first and second junction points, which is shown in Figure 3.26.

  Then break the edge $\{x_3, x_4\}$ into two edges, i.e., delete $\{x_3, x_4\}$ and add $\{x_3, b_{num}\}$ and $\{b_{num}, x_4\}$. Also, add edges $\{x_2, a_{num}\}$ and $\{a_{num}, b_{num}\}$.

  Again, all the x are node numbers of tubes in this group of consecutive merges at adjacent sides and they can be found.

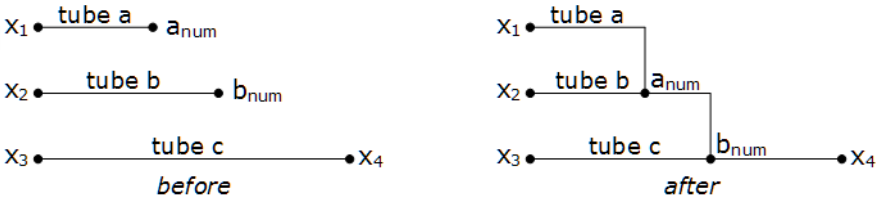


Figure 3.26: Nodes and edges for consecutive merges

- For merge/shift/merge $a \rightarrow b \rightarrow c \rightarrow d$, set nodes a_{num} and b_{num} to the first and second junction points, which is shown in Figure 3.27.

  Delete edge $\{x_2, b_{num}\}$, add $\{x_2, a_{num}\}$ and $\{a_{num}, x_5\}$.

Delete edge $\{x_4, x_6\}$, add $\{x_4, b_{num}\}$ and $\{b_{num}, x_6\}$.

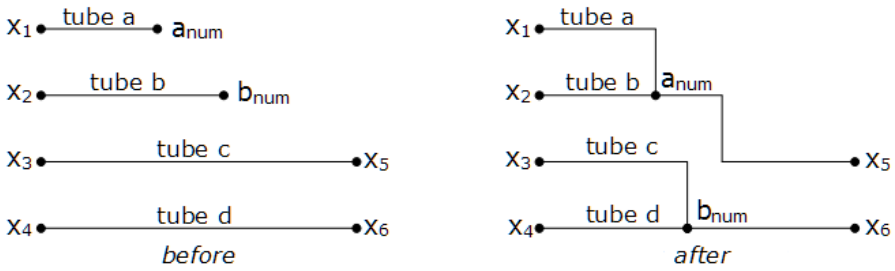


Figure 3.27: Nodes and edges for merge/shift/merge

With all the rules of assigning nodes and adding edges for the graph of a solution, one can store the graph by its incidence matrix, which will be discussed later.

Note that the total number of nodes in the graph of a solution never changed and it is always $N_1 + N_2 + \cdots + N_S$, which makes it convenient to build the incidence matrix.

3.5.2 Isolated loop and dead end

A tubular network is physically valid if and only if there is no isolated loop or dead end, which are defined below.

An **isolated loop** is a group of self-connected tubes with no connection to other tubes, and it may or may not contain an inlet or outlet. Two examples are shown in Figure 3.28.

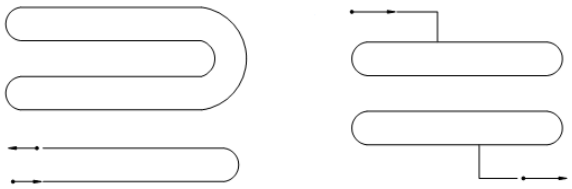


Figure 3.28: Two examples of isolated loop

In graph theory, a **connected component** of an undirected graph G is a subgraph G_1 in which any two vertices are connected to each other by paths, and which is not a proper subset of any other subgraphs of G.

Therefore, if the graph G constructed from a full solution has more than one connected component, then the solution is infeasible and should be discarded. An example of a graph with two connected components is provided in Figure 3.29.

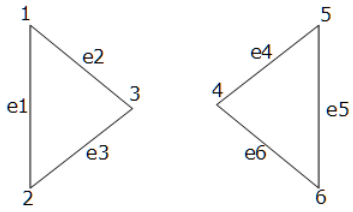


Figure 3.29: A graph with two components

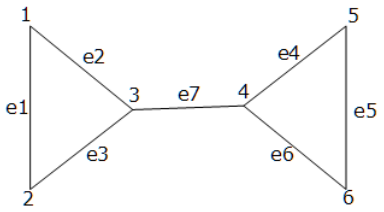


Figure 3.30: A graph with a bridge

A **dead end** is a group of self-connected tubes with only one path to the other part of the network. An example is shown in Figure 3.31 where the dead end is enclosed, by red dashed lines.

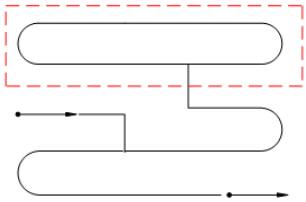


Figure 3.31: An example of dead end

In graph theory, a **cycle** is a circuit in which no vertex except the first (which is also the last) appears more than once. A **bridge** in an undirected graph is an edge whose removal increases the number of connected components of the graph. Most importantly, a bridge does not lie in any cycle in a graph. An example of a graph with a bridge is shown in Figure 3.30 in which edge e_7 is a bridge.

Note that when checking the existence of a bridge, the edge $\{inlet, outlet\}$ must be added to the constructed graph. Otherwise, many edges would be mistakenly identified as bridges. An example is shown in Figure 3.32. If the edge $\{v_1, v_6\}$ connecting the inlet and outlet is not added to the graph, then every edge in it is a bridge, which is misleading.

To algorithmically determine if the number of connected components is greater than 1 and determine if there is any bridge in the graph, some tools and theorems in graph theory are needed.

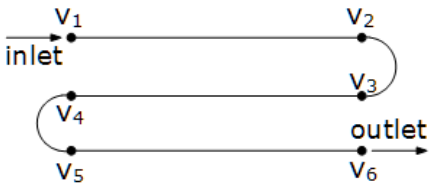


Figure 3.32: Inlet and outlet not connected by an edge

Definition 3.6 (Incidence matrix)**.** Let G be a graph with n vertices, m edges and without self-loops. The incidence matrix A of G is an $m \times n$ matrix whose n columns correspond to the n vertices and the m rows correspond to the m edges such that

$$a_{ij} = \begin{cases} 1, & \text{if } i\text{th edge is incident to } j\text{th vertex} \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

It is also called vertex-edge incidence matrix and is denoted by $A(G)$.

Remark. The matrix A has been defined over a field, Galois field modulo 2 or $GF(2)$, that is, the set $\{0,1\}$ with operation addition modulo 2 written as "+" such that $0+0=0$, $0+1=1$, $1+1=0$ and multiplication modulo 2 written as "$\cdot$" such that $0\cdot 0=0$, $0\cdot 1=0$, $1\cdot 1=1$.

The incidence matrices of the graphs shown in Figures 3.29 and 3.30 can be written as

$$A_1 = \begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \end{array}, \quad A_2 = \begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array} \tag{3.2}$$

The following theorem provides the basis of the algorithm of checking if a graph has more than one connected components.

Theorem 3.7. *Let $A(G)$ be an incidence matrix of a graph G with n vertices and m edges. If $rank(A(G)) < n-1$, then the graph G must have at least two connected components.*

Proof. Let f be a *locally constant* function on the vertices of G, i.e., for any two adjacent vertices $i \neq j$ we have $f(i) = f(j)$. Let $C = \{f : f \text{ is locally constant } \}$ be the space of functions which are locally constant. Then, by the definition of the incidence matrix A it is clear that $C \subset ker(A)$. Conversely, if $f \in ker(A)$ then clearly $f(i) = f(j)$ for any two adjacent vertices i, j and thus $f \in C$ and so $C = ker(A)$.

Now, suppose that $rank(A) < n-1$ so that $dim(ker(A)) \geq 2$. Then there are two nonzero $f, g \in ker(A)$ with $f \neq \lambda g$ for any constant λ. Choose a vertex i with $f(i) \neq 0$ and set $\lambda = g(i)/f(i)$. Then we have $h = g - \lambda f \neq 0$ (the zero function) but $h(i) = 0$ and $h \in ker(A) = C$. This means that h takes on at least two distinct values. Let $G_1 = h^{-1}(0)$ and $G_2 = G \backslash G_1$. The two sets G_1 and G_2 are disconnected in G (since h is locally constant but takes on different values on G_1 and G_2) and thus G is disconnected with at least two components. □

Returning to the two incidence matrices in Equation 3.2 and the corresponding graphs, by computing the rank with modulo 2 arithmetic, we have that the ranks of A_1 and A_2 are respectively 4 and 5, indicating that the graph at the left is disconnected and the one at the right is connected.

Now suppose that the graph G constructed from solution S_i has n vertices and m edges, the following steps are used to check if the graph G is disconnected, i.e., if it has more than one connected components.

Step 1. If $m \leq n - 2$, it can be concluded, without computation of the rank, that the graph is disconnected. Discard solution S_i and continue to check the feasibility of solution S_{i+1}.

Step 2. If $m \geq n - 1$, compute the rank of the incidence matrix $A(G)$ using modulo 2 arithmetic. If $rank(A(G)) < n - 1$, discard solution S_i and continue to check the feasibility of solution S_{i+1}; Otherwise, the graph G is connected, indicating that solution S has no isolated loop.

The following theorem will be used later, whose proof is similar to that of Theorem 3.7 and thus omitted here.

Theorem 3.8. *If G is a disconnected graph with k connected components, then the rank of its incidence matrix $A(G)$ is $n - k$.*

A connected graph may still have a bridge in it. The following tools and theorems are needed to build the algorithm of checking the existence of a bridge in a graph.

Definition 3.9 (Cycle matrix)**.** Let G be a graph with m edges and q different cycles in G. The cycle matrix B of G is a $q \times m$ matrix whose m columns correspond to the m edges and the q rows correspond to the q cycles such that

$$b_{ij} = \begin{cases} 1, & \text{if } i\text{th cycle contains } j\text{th edge} \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

An example in provided below.

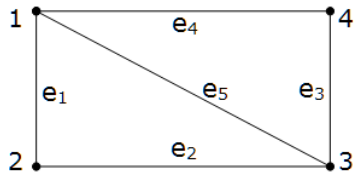


Figure 3.33: An example graph illustrating cycles

The graph in the figure above has three different cycles: $Z_1 = \{e_1, e_2, e_5\}$, $Z_2 = \{e_3, e_4, e_5\}$, $Z_1 = \{e_1, e_2, e_3, se_4\}$. The cycle matrix can be written as follows.

$$B = \begin{array}{c} \\ Z_1 \\ Z_2 \\ Z_3 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array} \tag{3.4}$$

Note that adding the first two rows by modulo 2 arithmetic gives the third row, which means the first and second cycles can form the third cycle by linear combination.

Definition 3.10 (Basis cycle)**.** A basis cycle in a graph is a cycle that cannot be obtained by linear combination of other cycles in the graph. All the cycles in a graph can be obtained by the set of basis cycles in it.

Obviously, the rank of a cycle matrix is the number of basis cycles. For instance, the graph in Figure 3.33 has two basis cycles Z_1 and Z_2, and the rank of its cycle matrix is 2.

Theorem 3.11. *Suppose a graph G has n vertices, m edges and k connected components, then the number of basis cycles in graph G is $m - n + k$. If B is a cycle matrix of graph G then $rank(B) = m - n + k$.*

Proof. This theorem comes from *Corollary* 4.6.7 in [14]. □

A column of all zeros corresponds to a *bridge* which does not belong to any cycle. For example, in Figure 3.30 there are two cycles $Z_1 = \{e_1, e_2, e_3\}$ and $Z_1 = \{e_4, e_5, e_6\}$ and the cycle matrix can be written as follows.

$$B = \begin{array}{c} \\ Z_1 \\ Z_2 \end{array} \begin{array}{c} \begin{array}{ccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \end{array} \\ \left[\begin{array}{ccccccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array} \right] \end{array} \tag{3.5}$$

Note that the last column has only zeros, indicating edge e_7 does not belong to any cycle, i.e., it is a bridge.

Now it is clear that one can determine if there is a bridge in a graph by checking if there is a column of all zeros in a cycle matrix. One way is to build the cycle matrix of a graph constructed from a solution, which is not difficult. A better way is to convert this problem to checking the incidence matrix of the graph.

Theorem 3.12. *If G is a graph without self-loops, with incidence matrix A and cycle matrix B whose columns are arranged using the same order of edges, then every row of B is orthogonal to every column of A, i.e., $A^T B^T = BA \equiv 0$.*

Proof. For any vertex v_i and any cycle Z_j in G, either $v_i \in Z_j$ or $v_i \notin Z_j$.

In the case $v_i \notin Z_j$ there is no edge of Z_j which is incident on v_i and in the case $v_i \in Z_j$ there are exactly two edges of Z_j which are incident on v_i.

Consider the ith column of A and the jth row of B. Since the edges are arranged in the same order, the rth entries of the column and row are both nonzero if and only if the edge e_r is incident on the ith vertex v_i and is also in the jth cycle Z_j.

We have $[A^T B^T]_{ij} = \Sigma[A^T]_{ir}[B^T]_{rj} = \Sigma[A]_{ri}[B]_{jr} == \Sigma a_{ri} b_{jr}$.

For each e_r of G, we have one of the following cases.

a. e_r is incident on v_i and $e_r \notin Z_j$. Here $a_{ri} = 1$, $b_{jr} = 0$.

b. e_r is not incident on v_i and $e_r \in Z_j$. Here $a_{ri} = 0$, $b_{jr} = 1$.

c. e_r is not incident on v_i and $e_r \notin Z_j$. Here $a_{ri} = 0$, $b_{jr} = 0$.

All these cases imply that the ith vertex is not in the jth cycle Z_j and we have $[A^T B^T]_{ij} = 0 \equiv 0 \pmod 2$.

d. e_r is incident on v_i and $e_r \in Z_j$.

Here we have exactly two edges, say e_r and e_t incident on v_i so that $a_{ri} = 1$, $a_{ti} = 1$, $b_{jr} = 1$, $b_{jt} = 1$. Therefore, $[A^T B^T]_{ij} = \Sigma a_{ri} b_{jr} = 1 + 1 = 0 \pmod 2$. □

We illustrate the above theorem with the example in Figure 3.30.

$$BA = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 2 \end{bmatrix} \equiv 0 \pmod 2 \quad (3.6)$$

Theorem 3.13. *Let A and B respectively be an incidence matrix and a cycle matrix of graph G with n vertices and m edges and k connected components. The dimension of the null space of A^T is equal to the rank of B.*

Proof. By *Rank-Nullity Theorem*([22]), we have $dim(N(A^T)) = m - rank(A)$. By Theorem 3.8, $rank(A) = n - k$. Thus $dim(N(A^T)) = m - n + k = rank(B)$ by Theorem 3.11. □

Combining Theorem 3.12 and Theorem 3.13, it follows that the null space(mod 2) of A^T is the row space of B, which is also called the cycle space of graph G.

Recall that if there is a column of all zeros in a cycle matrix $B(G)$ then there is a bridge in G. Based upon this idea, the algorithm is described as follows.

Step 1. Compute a set of basis vectors using modulo 2 arithmetic for the null space of A^T. Denote the basis vectors as $b_1, b_2, \cdots, b_{m-n+k}$, which are column vectors.

Step 2. Form a matrix $C = [b_1 \ \ b_2 \ \ \cdots \ \ b_{m-n+k}]$ whose order is $n \times (m - n + k)$.

Step 3. If the matrix C contains a row of all zeros, there must be a bridge.

Remark. When computing the basis for the null space of A^T, Gaussian elimination is inevitable, which can also give the rank of the incidence matrix A. In other words, the number of basis vectors for $null(A^T) = dim(ker(A^T)) = m - rank(A^T) = m - rank(A)$, and so we can know rank(A). Therefore, checking isolated group can be done simultaneously with checking dead end.

An example: The matrices formed by the basis vectors of the null spaces of A_1^T and A_2^T in Equation 3.2 are respectively

$$C_1 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \quad C_2 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}. \tag{3.7}$$

In matrix C_1, there is no row of all zeros so there is no bridge in the graph in Figure 3.29. In matrix C_2, the 4th row has all zeros so edge e_4 is a bridge in the graph in Figure 3.30.

3.6 9-4-9 problem

In this section we apply all the algorithms for building solutions at each side and verifying feasibility to the 9-4-9 problem which was introduced in Section 3.5.1.

Step 1. We first use *Unified Minimum Degree Matching Algorithm* together with *Depth-first Search* to enumerate all the solutions at interface S_2.

All the following steps are in the loop of i which indexes the solutions at S_2.

Step 2. Based on the ith solutions at side S_2, enumerate all the solutions at side S_3. Since solutions at side S_2 cannot affect the connection at S_3 and they have the same interface, these two sides have the same set of connection solutions.

All the following steps are also in the loop of j which indexes the solutions at S_3.

Step 3. Based on ith solution at S_2 and jth solution at S_3, use *Minimum Degree Matching Algorithm* together with *Depth-first Search* to enumerate all the solutions at S_1.

Whether the solutions at S_3, which is not adjacent to S_1, affect the connections at S_1 depends on the solutions at S_2. For the moment, we do not propagate constraints caused by solutions at non-adjacent sides.

Solutions at side S_2 do affect the connections at side S_1.

For example, the first solution at S_2 has one endcap 9&6 and three simple merges: 8→5, 7→4, 3→2.

Propagating constraints to extreme end S_1, these pairs cannot be connected at S_1: {9,6}, {8,5}, {7,4}, {3,2}.

The following step is also in the loop of k which indexes the solutions at S_1.

Step 4. Based on ith solution at S_2, $j$the solution at S_3, kth solution at S_1, enumerate all the solution at S_4.

Step 5. Check feasibility of the full solution composed of the ith, jth, kth, lth solutions respectively at S_1, S_2, S_3, S_4.

If feasible, store this full solution; Otherwise, discard it and proceed.

A few steps are demonstrated below:

- By enumeration, there are 22 connection solutions at both side S_2 and S_3.
- Based on the first solution at S_2 and the first solution at S_3, i.e., $i = 1$ and $j = 1$, there are 4 connection solutions at S_1.
- Given $i = 1$, $j = 1$ and $k = 1$, there are 4 connection solutions at S_4, which give four full potential feasible solutions. Verify the feasibility of each solution.
- Given $i = 1$, $j = 1$ and $k = 2$, there are again 4 connection solutions at S_4, which give another four full potential feasible solutions. Verify the feasibility of each solution.
- Continue the enumeration.

The 3D images of two solutions are provided below:

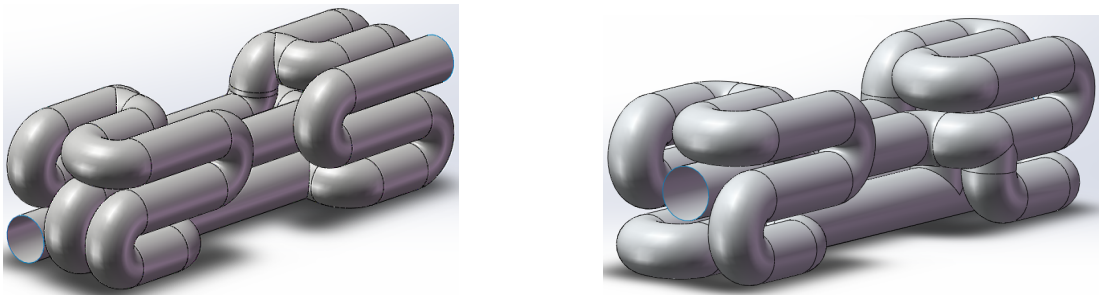


Figure 3.34: 3D images of two solutions to the 9-4-9 problem

This set of algorithms of enumerating all the feasible solutions based upon a given circle-packing in a prescribed region has been run on this 9-4-9 problem. It turns out that there are a total number of 6798 feasible solutions, which is a huge number for such a small sample problem.

As for the inlet and outlet of each tubular network generated by this set of algorithms, they are not specified individually in the form where tube a is the inlet and tube b is the outlet. Instead, the algorithm only designate a pair (a, b) as the inlet and outlet, which are exchangeable. Therefore, the number of feasible solutions should be doubled for asymmetric regions.

The region of the 9-4-9 problem happens to be symmetric because the two blocks B_1 and B_3 at two ends are exactly the same, show in Figure 3.22. However, if one of them is different, then this problem would have $6798 \times 2 = 13596$ feasible solutions.

3.7 Summary

In this chapter we have presented a series of algorithms for building connection solutions at each side of the discretized 3D region with tubes already packed and also presented a set of algorithms for verifying feasibility of a full solution.

For connecting the packed tubes, we only considered the situation in which all the tubes have the same radius. This set of algorithms are also applicable to the problem in which tubes have two radii if we introduce a new physical element, i.e., a bend with different

radii at two ends. Then there are three basic connection operations: an endcap, a merge and a shift operation with 2 radii can be adjust accordingly, as shown below.

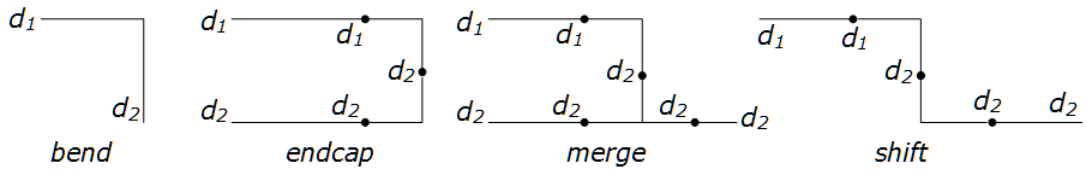
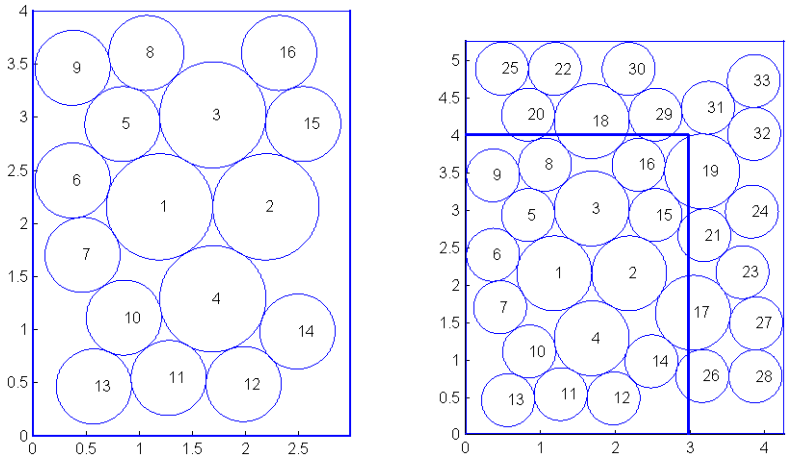


Figure 3.35: The new physical element and basic operations with 2 radii

Based on these three basic operations, the four connection operations can be constructed: an endcap, a simple merge, consecutive merges and merge/shift/merge. With these four operations, we can apply the same algorithms to generate feasible solutions. An example is provided below.

The 3D region is also a "saddle bag". The block at the middle of the saddle bag has the smaller cross section with only 16 circles packed. The two blocks at two ends have the bigger cross section with 17 more circles packed based on the circle-packing in the smaller cross section.



(a) Small cross section (b) Large cross section

Figure 3.36: Cross sections of blocks packed with tubes with two radii

Based on this set of circle-packing, one of many feasible solutions is provided below: **Connection at side S_1.**

- Endcaps: 24 & 19, 28 & 26, 33 & 32, 31 & 29, 25 & 20, 22 & 18, 21 & 23, 27 & 17, 16 & 3, 15& 2, 14 & 4, 12 & 11, 13 & 10, 7 & 6, 1 & 5, 8 & 9.

Connection at side S_2.

- Endcaps: 22 & 25, 33 & 31, 23 & 17, 24 & 21, 28 & 27, 30 & 18.
- Simple merges: 20→8, 29→16, 26→14.
- Consecutive merges: 32→19→15.
- Merge/shift/merge: None.

Connection at side S_3.

- Endcaps: 28 & 26, 23 & 27, 24 & 21, 32 & 19, 33 & 31, 30 & 18, 25 & 22.
- Simple merges: 17→14, 29→16, 20→8.
- Consecutive merges: None.
- Merge/shift/merge: None.

Connection at side S_4.

- Endcaps: 24 & 19, 30 & 29, 25 & 20, 28 & 27, 22 & 18, 31 & 32, 26 & 14, 23 & 21, 17 & 2, 16 & 15, 12 & 4, 13 & 11, 10 & 7, 6 & 5, 1 & 3, 8 & 9.

Tube 30 at side S_1 and tube 33 at side S_2 are not connected and hence designated as the inlet and outlet of the entire tubular network.

Of course this set of algorithms can be applied to the problems with more than 2 diameters if no new connection operations will be introduced, i.e., use only an endcap, a simple merge, consecutive merges and merge/shift/merge.

Among all the solutions of a problem, the best one can be obtained by evaluating all the solutions and choosing the best one. However, for bigger problems with more tubes packed in a region with more interfaces, it is not practical to enumerate all the feasible solutions in that this algorithm has factorial time complexity.

Therefore, we need to explore other approaches.

Chapter 4

Genetic Algorithm Approach

In this chapter, we discuss a simplified genetic algorithm (GA) approach to obtain a good, and hopefully optimal tubular networks for a given circle-packing.

Usually there are two important GA operators that generate new offspring:

- A crossover operator that combines existing solutions into others.
- A mutation operator that alters existing solutions into others.

In a genetic algorithm, the fitness value of an individual (solution) is a measure of how good it is in terms of the objective function of the optimization problem. More details on GAs and their applications may be found in [24].

A tubular network solution is extremely unstable in the sense that a small change in any connection operation in an existing solution Sol_0 may result in a dramatic change in the altered solution Sol_1 in order for it to remain feasible. Sometimes Sol_0 and Sol_1 are completely different solutions. Therefore, no desired property can be preserved or combined. As such, it is very difficult to imagine a crossover operator in our problem.

A mutation operator is indeed applied and it must be capable of altering an existing solution, whether in one step or in multiple steps, into any other solution in the entire solution space of the problem.

Here are the basic steps for the mutation-only genetic algorithm developed for our tubular network problem, each of which will be explained in the next few sections.

Step 1. Randomly generate N_0 solutions as the initial population of this genetic algorithm and evaluate the fitness of each individual in the population.

Step 2. Use *fitness proportionate selection* with mutation probability P_m to select $N_m = P_m \times N_0$ individuals to which the mutation operator will be applied.

Step 3. Apply the mutation operator to the N_m selected individuals to obtain N_m new individuals and evaluate the fitness of each of them.

Step 4. Now the population size is $N_0 + N_m$. Probabilistically eliminate N_m solutions according to fitness values to keep the population size fixed at N_0.

Step 5. Repeat Steps 2 to 4 until the termination condition is satisfied. Choose the individual with the highest fitness value that ever appeared in all generations.

Note that the encoding of a solution is the solution itself, i.e., the list of all types of connection operations and the involved tubes at each side.

4.1 Initial population and fitness value

Recall that the algorithms for generating connection solutions at each side of the discretized region are combined with depth-first search in order to enumerate all the potential solutions. In this case, adjacent solutions in the full set may be very similar to each other, e.g., differ in only two endcaps, because they were produced in a certain order.

However, similarity of solutions in the population reduces its diversity, which might adversely impact the performance of genetic algorithm. One way to increase the diversity of the initial population is to randomly generate connection solutions on each side.

Randomly generating a connection solution at an extreme end is straightforward: randomly choose one whenever there are multiple options.

Randomly generating a connection solution at an interface is somewhat more complicated because the length of each raw solution might be different. The following steps are applied in order to obtain a solution of random length.

Step 1. Randomly make connections whenever there are multiple options until each boundary circle has been connected to exactly one circle. Denote this raw solution by M_0 which contains n_{min} boundary circles.

Step 2. Use depth-first search to find out the longest possible raw solution based on this shortest solution M_0. Denote the max length by n_{max}.

Step 3. Randomly choose an even integer n_r within the interval $[n_{min}, n_{max}]$.

Step 4. Randomly generate a raw solution with length n_r.

In the problem in Figure 3.21, we have $M_0 = \{\{5,1\},\{8,2\}\}$. It is easy to determine the longest raw solution based on M_0 is $M_{max} = \{\{5,1\},\{8,2\},\{1,3\},\{2,4\},\{3,6\},\{4,7\}\}$ which represents two groups of merge/shift/merge operations: 5→1→3→6 and 8→2→4→7.

Randomly choose an even integer within the interval $[4,12]$, e.g., 8, then randomly generate a raw solution with length 8, i.e., 4 pairs. An example would be $M = \{\{5,1\},\{8,2\},\{1,3\},\{3,6\}\}$, which represents a group of merge/shift/merge 5→1→3→6 and an endcap 8&2.

Evaluation of each solution can be customized depending on the specific application of the tubular networks. Note that it is possible that no feasible solution can be constructed based on an existing solution with the mutation operator already applied. In this case, we still consider it as a new "solution" whose fitness value is *zero*, regardless of the evaluation function.

4.2 Selection and mutation operator

4.2.1 Roulette-wheel selection

The roulette-wheel selection is fitness proportionate, in which an individual with a higher fitness value has a greater probability of getting selected.

Suppose each individual in the population has a fitness value f_i, $1 \le i \le N_0$ and the mutation probability is P_m. The steps for roulette-wheel selection are described below:

Step 1. Sum all the fitness values: $F = \sum_{i=1}^{N_0} f_i$.

Step 2. Obtain the selection probability of each individual: $w_i = f_i/F$.

Step 3. Calculate the sum of first i fitness values of the individuals: $S_i = \sum_{j=1}^{i} w_i$.

Step 4. Independently generate $N_m = P_m \times N_0$ random numbers t_i from a uniform distribution on the interval $[0, 1]$. For each t_i, $1 \leq i \leq N_0$, find the smallest S_k that is greater than or equal to t_i, and then select individual k.

An example is provided in Table 4.1 where the population size is 10.

Table 4.1: An example of roulette-wheel selection

Individual i	1	2	3	4	5	6	7	8	9	10
Fitness value f_i	29	27	23	19	17	14	12	10	8	5
Selection probability w_i	0.18	0.16	0.14	0.12	0.10	0.09	0.07	0.06	0.05	0.03
Sum S_i	0.18	0.34	0.48	0.60	0.70	0.79	0.86	0.92	0.97	1.00

Suppose the mutation probability is $P_m = 0.4$ and 4 random numbers have been independently generated: 0.35, 0.86, 0.63, 0.12. The corresponding selected individuals are respectively 3, 7, 5, 1.

4.2.2 Mutation operator

The mutation operator in this algorithm must be capable of altering an existing solution, whether in one step or in multiple steps, into any other solution in the entire solution space of the problem.

Suppose the discretized region has a total number of S sides. The steps of mutating a full solution of the entire connection problem are as follows.

Step 1. Randomly generate a side number s within the interval $[1, S]$. Recall that side numbers 1 and S represent the two extreme ends and the other side numbers represent the interfaces.

Step 2. Let N_t be the number of tubes at an extreme end or the maximum number of circles in the two cross sections contained in an interface. For example, in the 9-4-9 problem, N_t is 9 at all four sides.

Randomly generate an integer T within the interval $[1, N_t]$ and find which group G of connection tube T is involved in, e.g., an endcap, a group of consecutive

merges. Mutate this group G: First remove all the connections and propagated constraints at the chosen side s except for G and then mutate tube T in one of the following three ways in order to form a new group of connections:

- Replace T with another tube, which leads to a new valid group of connections.
- Add a tube to group G to form a longer group of connections, e.g., from an endcap to consecutive merges.
- Delete a tube from group G to form a shorter one, e.g., from consecutive merges to a simple merge.

Restriction: At an interface, all these three ways of mutating tube T cannot lead to the situation where another boundary circle has a connectivity degree of zero.

The mutation of tube T in different cases and one common rule that applies to every case are described below:

Recall that a "1C" circle represents a first-layer common circle, a "1B" circle represents a second-layer boundary circle, further explanation and the definitions of "2C" and "2B" circles were discussed in Section 3.4.1.

Common rule. If there exists a tube Q that has the same layer-rank as that of T and that is available to all the tubes that are connected to tube T in the found group of connections, then T can be replaced by Q as a mutation. (Note that all the other connections and propagated constraints are now removed so that any tube with the properties described above can replace T.) For example, suppose that tube T is found in a group of consecutive merges: $X \rightarrow T \rightarrow P$. If there exists a 1$C$ circle Q that is available to both X and P, then tube T can be replaced by Q to form a new group of consecutive merges: $X \rightarrow Q \rightarrow P$.

All the possible mutations *in addition to* this common rule in every case are described below, some of which might be inapplicable depending on the specific T and G. Note that in some cases applying the common rule is the only possible mutation.

Case 1. Side s is an extreme end, i.e., $s = 1$ or $s = S$. If tube T is an inlet or outlet, put T in an endcap with one of its available tubes.

In the following cases side s is an interface, i.e., $2 \leq s \leq S - 1$.

Case 2. Tube T is in an endcap with a partner P.

2.1 If T is a $1B$ circle and it has an available tube Q which is an $1C$ circle, add Q to the endcap to form a group of consecutive merges: $P{\rightarrow}T{\rightarrow}Q$.

2.2 If P is also a $1B$ circle and there is an $1C$ circle Q available to it, replace T with Q and the endcap becomes a simple merge: $P{\rightarrow}Q$.

Case 3. Tube T is in a simple merge: $P{\rightarrow}T$. If there exists a boundary circle Q available to P, replace T with Q and the simple merge now becomes an endcap: $P\&Q$.

Case 4. Tube T is in a simple merge: $T{\rightarrow}P$. If there exists a boundary circle Q available to T, add Q to the simple merge to form the consecutive merges: $Q{\rightarrow}T{\rightarrow}P$.

Case 5. Tube T is in a group of consecutive merges: $X{\rightarrow}P{\rightarrow}T$.

5.1 Delete T from the group of consecutive merges, leading to an endcap: $X{\rightarrow}P$.

5.2 If there exists a common circle Q available to T, add Q to the group and form a group of merge/shift/merge: $X{\rightarrow}P{\rightarrow}T{\rightarrow}Q$.

Case 6. Tube T is in a group of consecutive merges: $X{\rightarrow}T{\rightarrow}P$.

Case 7. Tube T is in a group of consecutive merges: $T{\rightarrow}X{\rightarrow}P$. Break $T{\rightarrow}X$ and obtain an endcap $X{\rightarrow}P$. Further, if there exists another tube Q available to T, boundary or common circle, connected Q and T to obtain either another endcap $Q{\rightarrow}T$ or a simple merge $T{\rightarrow}Q$.

Case 8. Tube T is in a group of consecutive merges: $X{\rightarrow}Y{\rightarrow}P{\rightarrow}T$. Delete T and obtain a group of consecutive merges $X{\rightarrow}Y{\rightarrow}P$.

Case 9. Tube T is in a group of consecutive merges: $X{\rightarrow}Y{\rightarrow}T{\rightarrow}P$.

Case 10. Tube T is in a group of consecutive merges: $X{\rightarrow}T{\rightarrow}Y{\rightarrow}P$.

Case 11. Tube T is in a group of consecutive merges: $T{\rightarrow}X{\rightarrow}Y{\rightarrow}P$.

Identify and count the number of *applicable* mutations of a group G of connection, and then randomly choose one.

After the group G of connections has been mutated to a new group G_{new}, we can proceed to the next steps of mutating a full solution.

Step 3. Based on G_{new}, make connections on side s and store them into M_{fix}, using its corresponding algorithm, until there are multiple options of choosing the tube to be connected, i.e., connect only all the fixed pairs which have to be connected.

Note that a tube with a connectivity degree of 1 does not necessarily belong to a fixed pair because it is possible that another tube has the same unique available tube, giving rise to multiple options.

For example, if tube a has only one available tube c and so does tube b, then neither $\{a, c\}$ nor $\{b, c\}$ is a fixed pair.

Step 4. Propagate the constraints caused by M_{fix} to the adjacent sides of side s that might be affected. If the existing connection solution M_{adj} at an adjacent side conflicts with M_{fix}, i.e., they form at least an isolated loop or an dead end, then remove it and randomly regenerate a new M_{adj} based upon M_{fix}.

Step 5. Go back to complete the connection solution at side s with all the constraints propagated from its adjacent sides.

Step 6. Verify the feasibility of the full solution obtained in the previous steps. If feasible, accept it as a new individual and the process of mutation of a full solution should stop here.

Step 7. If the new full solution is not feasible, regenerate a new M_{adj} for each adjacent side that might be affected based upon M_{fix} and repeat Steps 5 and 6 for a certain number of times, e.g., 10 times. If still no feasible full solution can be found, then create a dummy full solution as the new individual and assign a fitness value of zero to it. That dummy solution will definitely be eliminated, which will be discussed later.

We repeat Steps 5 and 6 because there is no reason to discard the mutated new group G_{new} of connections based on one pass of randomly generating a connection solution. However, if after a good number of times of trials still no feasible solution can be found, then it might be that G_{new} is incompatible with the connection solutions at all the other sides.

The following is an example of mutating a full solution.

Suppose that in the 9-4-9 problem, the solution in Section 3.5.1 is selected for mutation.

- Randomly choose a side number within the interval $[1, 4]$: we have $s = 2$ which represents an interface.
- Randomly choose a number within the interval $[1, 9]$, we have $T = 5$.

  In the connection solution at side 2, tube 5 is in a group of consecutive merges: 9→8→5.
- There are three possible mutations: the common rule, Case **5.1** and Case **5.2**.

  As for the common rule, tube 5 is the only common circle available to tube 8 and hence it is inapplicable in this specific situation.

  Case **5.1**, in which tube T will be deleted to form an endcap, is always applicable.

  Case **5.2**, in which another tube will be added, is applicable here, and there are two choices: tube 2 and tube 4.

  Recall the restriction which states that mutation of tube T cannot lead to the situation where another boundary circle has zero connectivity degree. Therefore tube 4 cannot be added to the group of consecutive merges because tube 7 would have zero degree if done so.
- Randomly choose one between the two applicable cases and apply the mutation to obtain a new group of connections: we have a new group G_{new}: 9→8→5→2.
- Connect all the fixed pairs at side 2 based on G_{new} and we have $M_{fix} = \{\{G_{new}\}, \{3, 6\}, \{7, 4\}\}$.
- The adjacent side(s) of side 2 that might be affected is side 1 which is an extreme end. Also, there happens to be no conflict between M_{fix} at side 2 and the existing solution at side 1.
- Go back to side 2 and complete the connection solution based on M_{fix} and the solution at side 1. In fact, M_{fix} has already covered all the boundary circles.
- Check feasibility and repeat the steps if necessary.

Apply mutation to all the $N_m = P_m \times N_0$ selected individuals and add the new individuals to the current population.

4.3 Elimination and termination

After mutation, the size of the population will be $N_0 + N_m$. In order to keep the size fixed, N_m individuals must be eliminated probabilistically. The steps are as follows:

Step 1. Rank all the individuals by fitness values in descending order.

Step 2. Let N_{f0} denote the number of individuals with zero fitness value and eliminate all the individuals with zero fitness value. This is because individuals with zero fitness values must be dummy solutions produced due to the failure of mutation. (Note that a feasible solution must have a positive fitness value.)

Step 3. Start with the individual with the lowest fitness value, i.e., the fth where $f = N_0 + N_m - N_{f0}$, and choose a random integer I_r. If I_r is even then eliminate the fth individual; Otherwise apply the same procedure to the $(f-1)$th individual and repeat until N_m individuals have been eliminated.

Note that if the size of the population is still larger than N_0 after one loop of all the individuals, then go back to the individual with the lowest fitness value.

Note that in this method of eliminating individuals, the one with the highest fitness value might be eliminated with a very low probability, indicating the last generation may not contain the best individual that ever appeared in the process. Therefore, we have to keep track of the best individual.

The reason why we do not truncate all the N_m worst ones is that individuals with low fitness value might become one with a high fitness after more mutations. Also, probabilistic elimination helps maintaining the diversity of the population, which positively affects the performance of the genetic algorithm.

There are various criteria of termination and some of them are listed below:

Type 1. Terminate after a certain number of generations.

Type 2. Terminate after a certain period of CPU time.

Type 3. Terminate when the difference in fitness values between the best and worst has reached a certain value.

Type 4. Terminate when there has been no improvement in the last certain number of generations.

Appropriate termination criteria should be chosen depending on the specific application and its constraints.

4.4 Results and summary

In this section we apply this genetic algorithm to the 9-4-9 problem.

First, we randomly generated 20 feasible solutions as the initial population. Then we use the internal volume of all the tubes in the network as the fitness value of each solution. When computing the internal volume the interference problem should be taken into consideration, which is illustrated below.

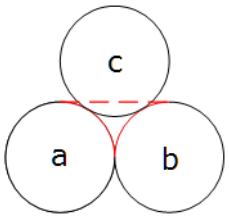


Figure 4.1: Interference caused by an endcap

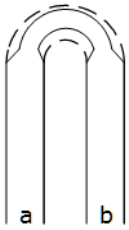


Figure 4.2: Narrowed endcap

In Figure 4.1, if tubes a and b are connected by an endcap then the cap would block the region enclosed by red lines, hence leading to the impossibility of extending tube c or making connections between tube c and other tubes. To solve this problem, we can use the strategy of narrowing down only the two 90-degree bends as shown in Figure 4.2.

Similarly, since all the tubes are tightly packed touching each other, a merge or a shift can also result in interference. Again, we can narrow down the physical elements that connect the tubes to reduce the interference. The internal volume of a tubular network can be easily computed and the details are omitted here.

The genetic algorithm was run for 20 generations with a mutation probability of 0.3 and the process of evolution of the population is shown in Figure 4.3. As shown, the average of the fitness values of the population is improving during the evolution process and the

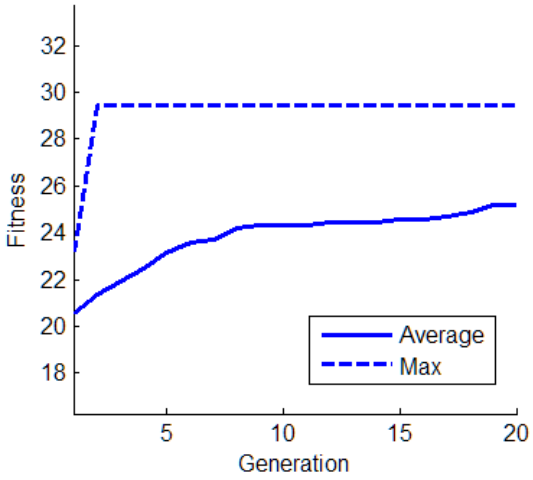


Figure 4.3: Highest and average fitness value of each generation

maximum possible fitness value is reached at the second generation. In this case, the internal volume of solutions only slightly deviate from each other because all the solutions differ only in the numbers of all types of connection operations. Further, solutions with the same number of each type of connection operation always have the same internal volume. One solution with the highest fitness value is given below:

Connection at side S_1. Endcaps: 3 & 6, 7 & 8, 1 & 2, 4 & 5.

Connection at side S_2.

- Endcaps: 9 & 6.
- Simple merges: 8→5, 7→4, 3→2.

Connection at side S_3.

- Endcaps: 9 & 6.
- Simple merges: 8→5, 7→4, 3→2.

Connection at side S_4. Endcaps: 3 & 6, 7 & 8, 1 & 4, 2 & 5.

The mutation-only genetic algorithm described in this chapter is able to search the entire solution space of the connection problem. Every aspect of this algorithm, including the selection strategy, elimination and termination condition, contributes to its efficacy. It is much more efficient that the impractical method of enumerating all feasible solutions.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis we have presented a series of algorithms for constructing optimal tubular networks in arbitrary regions in R^3. Given a 3D region, we first discretize it along a principal direction and obtain different blocks, each of which exhibits no variation in cross section. Then the problem has been divided into two parts:

1. Pack tubes into the blocks, which reduces to packing circles into the cross sections, approximated by polygons, of the blocks.
2. Connect the packed tubes with different operations and verify feasibility of each solution.

5.1.1 For circle-packing

Given a set of cross sections of the blocks, we first pack circles into an interior region which is common to several or all cross sections and the remainder of the unpacked region are packed afterwards. We have also developed a set of novel circle-packing algorithms for arbitrary polygons.

In chapter 2 we have presented various algorithms for packing different-sized circles into an arbitrary polygon and the ultimate circle-packing possesses the following two features:

a). Larger circles are situated primarily in the interior of the region.

b). As much of the remaining empty space as possible is along the boundary.

We described the GGL algorithm of packing circles into a rectangular region and then extended this algorithm to packing circles into an arbitrary polygon. The GGL-based algorithm starts packing circles at corners and develops both upwards and to the right. Although a circle-packing generated by the GGL-based algorithm does not possess the desired features, the method of placing circles by position numbers can still be employed in other algorithms we developed.

In an attempt to have larger circles packed primarily in the interior regions, we designed the Reversed-GGL algorithm which starts packing circles at the interior of the region and proceeds towards the boundary. However, the Reversed-GGL algorithm has a limitation– sometimes the circle-packing cannot reach very narrow corners, which is due to the rule that only two-circle packing is allowed. In order to eliminate this limitation, we introduced the rule of one-circle packing which allows a candidate circle to be packed tangent to only one previously packed circle. As expected, the circle-packing produced by the Reversed-GGL algorithm with one-circle packing can reach very narrow corners. A disadvantage of one-circle packing is that it is less compact than two-circle packing, which means probabilistically much more iterations are needed to generate a circle-packing as good as the ones generated by algorithms employing only two-circle packing.

The Reversed-GGL algorithm with one-circle packing still cannot restrict large circles in the central region of a polygon. Therefore, we imposed a set of additional constraints on the distance between the center of a large/medium circle and the boundary. With this set of constraints, all the large/medium circles are restricted to the central part of the polygon and thus the first desired feature is satisfied.

We have also developed the Hybrid Circle-packing algorithm to restrict large circles to the central region of a polygon, which is called hybrid circle-packing algorithm. It first uses the Reversed-GGL algorithm to fill the polygon with different-sized circles, then removes the outermost layer of circles of the circle-packing, and finally employs the GGL-based algorithm to pack circles between the boundary and the second outermost layer. This algorithm can indeed restrict large circles in the central region of polygon but it is complicated and it does not have control over the distance between large circles and the boundary.

In order to accelerate the process of obtaining a decent or optimal circle-packing, we have

designed the "jiggling" algorithm which imposes a fictitious force field that attracts all the small and medium-sized circles towards the "center of mass" of a given circle-packing. As a result, in addition to the acceleration of circle-packing, more empty space along the boundary is released, satisfying the second feature that is desired.

In summary, constraints on the distance between large circles and the boundary should be employed in every circle-packing algorithm. As well, jiggling should be employed in order to improve each packing. The various circle-packing algorithms developed in this thesis are listed below with the planar regions over which they are suitable:

1. Reversed-GGL algorithm: Only for polygons without needle-like regions.
2. Reversed-GGL algorithm with one-circle packing: For any polygon.
3. Hybrid circle-packing algorithm: Applicable to any polygon if there is no specific requirement on the distance between large circles and the boundary.

5.1.2 For connection

As for connecting the packed tubes in the discretized region, we only allowed three physical elements: a straight pipe, a 90-degree bend and a tee joint, from which four connection operations have been constructed: an endcap, a simple merge, consecutive merges and merge/shift/merge.

Using these four connection operations, we have designed a series of algorithms for enumerating all the feasible solutions and for randomly generating a subset of the entire solution space and choosing the best one. With the blocks of the 3D region, we introduced the concepts of "side", i.e., an extreme end of the 3D region or an interface of two blocks. Then we developed algorithms that allow us to generate a connection solution for each side separately, which composes a potentially feasible full solution for the entire connection problem.

For a connection solution at an extreme end, the problem has been converted to the one of finding maximum matchings in the graph constructed from the circle-packing. The *Minimum Degree Matching Algorithm* is an extremely efficient algorithm (linear time complexity) for generating matchings that are extremely close to a maximum matching of a graph, and in most cases it generates maximum matchings. If combined with *Depth-first Search*, it is capable of enumerating all the maximum matchings in a graph, which is desired.

For a connection solution at an interface, we have designed a novel algorithm which is called the *Unified Minimum Degree Matching Algorithm*, so named because all the four operations are dealt with in the same way. The main idea of this algorithm is similar to that of the *Minimum Degree Matching Algorithm*, i.e., repeatedly picking edges incident to nodes of current minimum non-zero connectivity degree. In this algorithm, we introduced two new concepts of "layer-rank" and "connectivity value" that made it possible for a tube to be connected to two other tubes. At the last step of this algorithm, a raw solution is interpreted into a meaningful connection solution. Again, if combined with *Depth-first Search*, this algorithm can enumerate all the possible connection solutions at an interface.

A potentially feasible full solution is composed of connection solutions at each side of the discretized 3D region. A feasible tubular network does not have any isolated loop or dead end, which involves the two concepts of "connected graph" and "bridge" in graph theory. Therefore, some aspects of graph theory were introduced in order to algorithmically verify the feasibility of each potential solution.

We constructed a graph based upon all the connection operations of the entire tubular network and stored it in the form of an incidence matrix. Building upon a series of theorems in graph matrix theory, we developed an algorithm that only needs to check the rank of the incidence matrix A to determine if the graph is disconnected or not. Also, we designed an algorithm that computes a set of basis vectors for the null space of A^T to determine if there is any bridge. When computing the basis for the null space of A^T, Gaussian elimination is inevitable, which can also give the rank of the incidence matrix A. Therefore, checking for the occurrence of both isolated groups and dead ends can be done simultaneously.

We applied all the algorithms to a sample problem, i.e., the 9-4-9 problem, and obtained 6798 feasible solutions by enumeration. This set of algorithms can also be applied to problems where tubes have different radii by introducing a new physical element: a bend with different radii at its two ends. Among all the solutions of a problem, the best one can be obtained by evaluating all the solutions and choosing the best one. However, for bigger problems with more tubes packed in a region with more interfaces, it is not practical to enumerate all the feasible solutions. Therefore, we designed a genetic algorithm with only a mutation operator that generates new solutions. The mutation operator in this genetic algorithm is capable of altering an existing solution to any other solution in the entire solution space of the connection problem and thus the algorithm can produce an optimal solution.

5.2 Future work

In this thesis, all the circle-packing algorithms are applicable only to regions enclosed by simple closed curves, i.e., regions without "holes". One possible extension is to adapt these algorithms to pack non-simple regions similar to the one shown below.

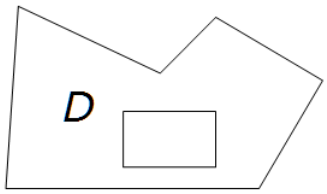


Figure 5.1: A non-simple region

In the process of obtaining a connection solution at each side, we only propagated constraints between adjacent sides to immediately avoid isolated loops and dead ends. It is desirable to propagate constraints between non-adjacent sides, which might accelerate the algorithm and hence save computational resources.

When determining the available tubes for connection of a tube, we only considered adjacent tubes. One may wish to consider connections between non-adjacent tubes by quantifying the interference with other tubes caused by those connections. Accordingly, the layer-ranks would be depending on availability for connection instead of adjacency. Another possible extension is to add more connection operations for problems with multiple radii. For example, one can allow an endcap that connects three tubes.

Recall that a merge or a shift operation is only allowed at interfaces. One may wish to allow merges or shifts to occur at or near the extreme ends. This would necessitate the modification of our existing algorithms. Moreover, a much larger number of feasible networks would result.

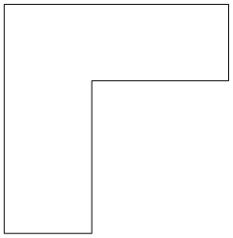


Figure 5.2: Front view of a 3D region with two principal directions

An assumption of this entire problem is that the prescribed 3D region always has one principal direction. A direction of research could be investigating the problem in which the 3D region has multiple principal directions.

References

[1] http://www.redneckpoolheater.com/.

[2] Bernardetta Addis, Marco Locatelli, and Fabio Schoen. Efficiently packing unequal disks in a circle. *Operations Research Letters*, 36(1):37–42, 2008.

[3] Bert Besser. Approximation bounds for minimum degree matching. *arXiv preprint arXiv:1408.0596*, 2014.

[4] Paul Bourke. Calculating the area and centroid of a polygon, 1988.

[5] Ignacio Castillo, Frank J Kampas, and János D Pintér. Solving circle packing problems by global optimization: numerical results and industrial applications. *European Journal of Operational Research*, 191(3):786–802, 2008.

[6] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[7] Kathryn A Dowsland. Optimising the palletisation of cylinders in cases. *Operations-Research-Spektrum*, 13(4):204–212, 1991.

[8] Herbert Edelsbrunner. *Algorithms in combinatorial geometry*, volume 10. Springer Science & Business Media, 1987.

[9] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[10] H.J. Fraser and J.A. George. Integrated container loading software for pulp and paper industry. *European journal of operational research : EJOR*, 77(3):466–474, 1994.

[11] Alan Frieze, AJ Radcliffe, and Stephen Suen. Analysis of a simple greedy matching algorithm on random cubic graphs. *Combinatorics, Probability and Computing*, 4(01):47–66, 1995.

[12] John A George, Jennifer M George, and Bruce W Lamar. Packing different-sized circles into a rectangular container. *European Journal of Operational Research*, 84(3):693–712, 1995.

[13] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[14] Jonathan L Gross and Jay Yellen. *Graph theory and its applications*. CRC press, 2nd edition, 2005.

[15] Mhand Hifi and Rym M'hallah. A literature review on circle and sphere packing problems: models and methodologies. *Advances in Operations Research*, 2009, 2009.

[16] Quan Zhang Huaiqing Wang, Wenqi Huang and Dongming Xu. An improved algorithm for the packing of unequal circles within a larger containing circle. *European Journal of Operational Research*, 141(2):440–453, 2002.

[17] W. Jiang, B. Kettlewell, T. Qiao, F. Mendivil, S. D. Peterson, and E. R. Vrscay. The barbeque pool heater: An algorithm to construct tubular networks that occupy arbitrary regions in r^3. *Applied Mathematics, Modeling and Computational Science - Canadian Applied and Industrial Mathematics Society Congress, Waterloo, ON, June 7-12*, 2015.

[18] W. Jiang, T. Qiao, F. Mendivil, S. D. Peterson, and E. R. Vrscay. Some novel circle-packing algorithms devised for the construction of tubular networks in r^3. *Applied Mathematics, Modeling and Computational Science - Canadian Applied and Industrial Mathematics Society Congress, Waterloo, ON, June 7-12*, 2015.

[19] Richard M Karp and Michael Sipser. Maximum matching in sparse random graphs. In *Foundations of Computer Science, 1981. SFCS'81. 22nd Annual Symposium on*, pages 364–375. IEEE, 1981.

[20] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196, 2012.

[21] Jakob Magun. Greeding matching algorithms, an experimental study. *Journal of Experimental Algorithmics (JEA)*, 3:6, 1998.

[22] Carl D Meyer. *Matrix analysis and applied linear algebra*. Siam, 2000.

[23] Silvio Micali and Vijay V Vazirani. An o (v— v— c— e—) algoithm for finding maximum matching in general graphs. In *Foundations of Computer Science, 1980., 21st Annual Symposium on*, pages 17–27. IEEE, 1980.

[24] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[25] Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 248–255. IEEE, 2004.

[26] Washington A Oliveira, Luiz L Salles Neto, Antônio C Moretti, and Ednei F Reis. Nonidentical circle packing problem: multiple disks installed in a rotating circular container. *arXiv preprint arXiv:1401.4952*, 2013.

[27] Péter Gábor Szabó, Mihaly Csaba Markót, Tibor Csendes, Eckard Specht, Leocadio G Casado, and Inmaculada García. *New approaches to circle packing in a square: with program codes*, volume 6. Springer Science & Business Media, 2007.