Packing 模型在城市绿化规划中树木优化运用

课程名称 数学实验

上课班级：992304-003

姓名	学院	年级	专业	学号	点名册序号
文红兵	计算机学院	2021 级	计算机科学与技术	20214590	57
张奎元	计算机学院	2021 级	计算机科学与技术	20214358	38
苟书祥	计算机学院	2021 级	计算机科学与技术	20214056	无（温罗生老师）

开 课 时 间 2022 至 2023 学年第 二 学期

重庆大学

数学建模校内竞赛论文

论文题目： Packing 模型在城市绿化规划中树木优化运用

组号：8155

成员：文红兵，张奎元，苟书祥

选题：城市绿化规划中树木的优化配置

姓名	学院	年级	专业	学号	联系电话	数学分析	高等代数	微积分	高等数学	线性代数	概率统计	数学实验	数学模型	CET4	CET6
文红兵	计算机学院	2021 级	计算机科学与技术	20214590	15310865763	无	无	无	94	85	89	无	无	503	无
张奎元	计算机学院	2021 级	计算机科学与技术	2021358	15039445584	无	无	无	92	81	96	无	无	556	455
苟书祥	计算机学院	2021 级	计算机科学与技术	20214056	18282779356	无	无	无	100	85	85	无	无	无	无

摘要

本文旨在建立一个优化配置模型，确定在给定土地上可以种植的最多树木数量或者最大树高，并满足题目给定的约束条件。在求解中通过设定必要的假设，将问题抽象为二维矩形面积下所能容纳圆形的最大数量。针对该问题，本文主要通过建立 Packing 模型进行求解。

针对问题一：在假设树高一致的前提下，通过题干建立关于树高相关函数，利用图像法求得树高的最小值，进而求得树木覆盖半径。在求得最小半径以后，采用“拟物”算法对 Packing 问题进行进一步求解。求解中运用的优化策略有：

(1)利用 sigmoid 函数将模型产生的非连续性函数转化为连续函数。

(2)对矩形土地进行合适的划分加速问题的求解。

得到最多可以种植 10000 棵树。

针对问题二：在问题一的基础上，结合 Packing 问题的求解思路，将建立的具体函数求解目标转化为求解新栽种树木之间“势能”加上“新栽种树木”与初始栽种树木之间“势能”之和是否可以为 0。

针对问题三：我们可以将问题建模为一个优化问题，其中目标是最大化树的高度，约束条件是每对树之间的距离不小于它们的占地面积的一半，运用合适算法对其求解。

针对问题四：由于土地面积确定，但是长和宽不定。通过对长和宽的遍历，在不同土地形状下，用拟物算法求解。其中树木占地半径与第一问保持一致。选取最大值作为最终求解结果，**得到最多可以种植 12 棵树**。。

关键词： 树木种植；优化配置模型；Packing 模型；最优解

1 问题重述

随着城市化进程的日益加快,人们对城市绿化问题的关注也在不断提高。城市绿化问题不仅关乎着群众生活幸福感的提升,还与城市样貌,环境保护等息息相关。通过建立合适的优化配置模型,可以帮助城市规划师和环保部门更科学地规划城市绿化,合理分配有限资源,进而提高城市绿化的质量和效益。

1.1 要求

本题要求建立一个优化配置模型,确定在给定土地上可以种植的最多树木数目,同时需要满足以下条件:

1. 每棵树占地 10 平方米且不能与其他树的占地重叠。
2. 树冠面积不能超出土地边界。(其中树冠覆盖面积同树高正相关,树的高度范围在 1-10 米之间)
3. 树的树干之间必须留出半径为 2.5 米的安全距离。
4. 所有树的高度尽量相同。
5. 每棵树的种植成本等于 10 倍树高(米)+10 元。

1.2 问题

问题一:通过建立合适的数学模型,确定在给定土地上可以种植的最多树木数目,同时满足上述所有条件。

问题二:假设在 500 米 ×500 米的土地上已经种植了一些树木,确定在已种植的树木基础上,还能种植多少树木。

问题三：假设在 500 米 ×500 米的土地上已经种植了一些树木，需要调整树木的高度，以最大化覆盖面积。

问题四：假设只有 300 平方米的土地可用，确定最多可种植多少棵树，以及如何种植才能达到最优解。

2 问题与背景分析

2.1 背景分析

城市绿化是指通过植树造林、建造草坪、种植花草等植物，对部分土地或者空间进行覆盖或者装点，改善城市环境、提升生态质量的一种行为。随着近年来工业化、城市化进程的加速，城市环境污染、生态破坏和人居舒适度下降等问题开始日趋严峻。因此，城市绿化建设在城市规划建设中变得尤为重要，城市绿化的主要目标是为人们提供良好的生态环境和居住条件，以改善城市居民的生活质量。

城市绿化规划需要考虑的内容主要有：土地面积、种植植物的数量和类型、空间分布等因素。其中，树木作为城市绿化的重要组成部分，不仅能够为城市中的动植物提供氧气、吸收二氧化碳，还有调节温度、保护水源、改善风景等重要作用。因此，在城市绿化规划中，合理配置树木资源对于提高绿化质量和效益至关重要。

本题要求求解在给定的 $500m \times 500m$ 土地上可以种植的最多树木数目，并满足多个约束条件。这些约束条件包括树木的占地面积、树冠覆盖面积、安全距离和树木高度的一致性等。此外，每棵树木的种植成本也是一个考虑因素。通过建立数学模型，我们可以进行优化配置，

以使城市绿化达到最佳效果。通过解决以上问题,旨在帮助城市规划师和环保部门进行科学的城市绿化规划决策,确保树木的最大利用和城市绿化的可持续发展。

2.2 问题分析

问题一分析: 解决该问题之前,我们已经假设每棵树的高度一致,根据冠幅与树高的关系可知,每棵树冠幅的面积也相同。在矩形面积一定的情况下,为了使所能栽种的数木数量达到最大,那么每棵树种植所需要的面积需要最小。因此解决该问题需要通过建立最小化面积模型求得单个树木种植所需的最小面积,然后建立 Packing 模型来求解在满足题干要求的基础上,矩形土地上所能容纳树木数量的最大值。

问题二分析: 在问题一的基础上,可以利用随机坐标生成模型在满足题干要求的基础上生成矩形土地上已种植树木的坐标。接着再利用 Packing 模型对可种植树木的数量进行进一步求解,其中树木半径取第一问求得的半径。

问题三分析: 首先利用随机坐标生成模型满足题干要求的基础上对已种植树木坐标进行求解。接着利用最大值求解模型对树木之间距离的最大值进行求解。在求得树木之间最大值之后,利用最小值求解模型求得树木种植距离的最大值。进而求得树木种植所需半径,利用树冠覆盖面积与树高的关系求得树高。

问题四分析: 在问题一的基础上,首先,要使得种植的树木数量最多,根据第一问,树木的占地半径需要为 2.5。面积为 $300m^2$ 的土地长和宽分别用 L 和 W 表示。L 和 W 为本题的变量,满足限制条件 $L \times W = 300$ 。因此需要依次遍历 L 和 W 的取值,运用第一问中提到的拟物算法,求得每种情况下种植树木数量的最大值,最后再对所有取最大值,解出最优方案。

3 模型假设

- 假设每棵树的占地面积为树冠覆盖面积和最小圈地面积之间的较大值：此处的最小圈地面积为 10 平方米（题干中已知），由于树冠覆盖面积与绿化树木的高度有关，且高度越高，树冠覆盖面积越大，不能保证树冠覆盖面积一定小于最小圈地面积，因此假设每棵树的占地面积为树冠覆盖面积和最小圈地面积之间的较大值。
- 假设树冠的形状是圆形：由于本题主要探究的问题是与占地面积以及树冠覆盖面积有关，而常见的绿化树木树冠多为球形、圆锥形、抛物线体型、以及圆柱体型等，当投影到二维平面上时，它们的投影大都可以近似看作圆形。
- 假设树干是圆柱形：日常生产生活中常见的树木均为圆柱形，这一假设符合常识。
- 假设树的位置可以用二维坐标表示：题目中给出的土地形状为矩形，方便直角坐标系的建立。坐标位置假设为树干圆心的位置，绿化植物大多为直立生长，可以看作是以树干为中心发散生长。
- 假设土地都是矩形土地：由于题目中出现了 500×500 平方米的土地，且生活中土地的划分也大多划分为矩阵形状，因而在问题中出现的土地面积均假设为矩形土地。
- 假设树的高度一致：题目条件说尽量相同，为了简化问题，我们假设树的高度一致。生活中林场规划树木高度一般差异也很小，且树木高度的测量本身也并非绝对精准，因此这一假设是合理的。

4 符号说明

符号	意义	单位
n	种植树木的数量	棵
x_i	树的横坐标	m
y_i	树的纵坐标	m
h	树的高度	m
A	树的覆盖面积	m^2
r	树的占地面积的半径	m
S	树的占地面积	m^2
p	树的冠幅	m
L	土地的长	m
W	土地的宽	m

5 模型建立与求解

5.1 问题一的模型建立与求解

- 首先对树高与冠幅进行数据拟合，保证树高的取值在 1-10 之间都有对应的冠幅。得到如图1：

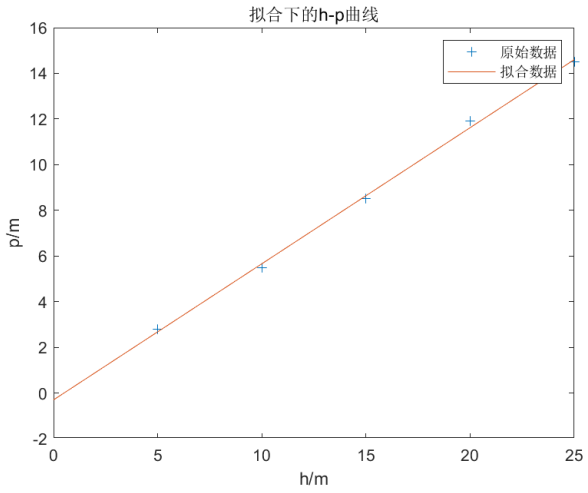

图 1: 冠幅与树高的关系

其中拟合检验参数如表1:

R^2	1
F	180.01
P	0

表 1: 拟合参数表

残差图如图2:

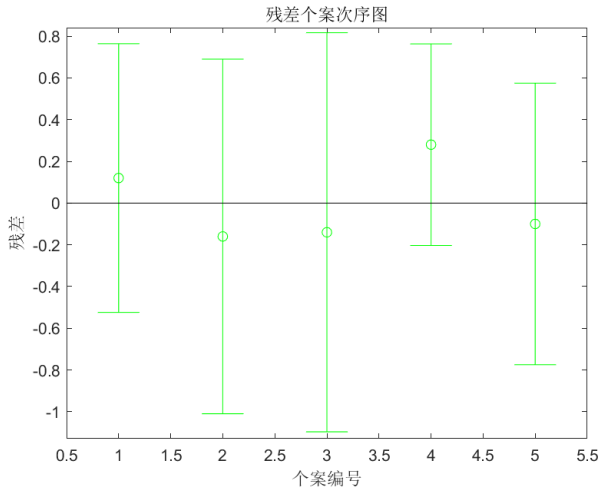

图 2: 冠幅与树高残差图

其中 R^2 的值为 1,解释程度较高,F 估计值为 180.01,远大于所设定置信度下的目标值,P 为 0,模型与被解释变量相关性较大。残差图正常,无异常点,此该回归模型合适。得到拟合公式:

$$p = -0.3 + 0.596h$$

- 要使在特定土地上的种植树的树木最大,根据假设,每棵树占地面积相同,因此使占地面积最小化就可以,也就是使每棵树的占地面积的半径最小化,得到如下目标优化:

$$\begin{aligned} &\min r \\ \text{s.t.}\quad &p = -0.3 + 0.596h \\ &A = \pi \times \left(\frac{p}{2}\right)^2 \\ &S = \max(10, A) \\ &r = \sqrt{\frac{S}{\pi}} \\ &r \geq 2.5 \\ &r \leq \frac{L}{2} \\ &r \leq \frac{W}{2} \\ &1 \leq h \leq 10 \end{aligned}$$

化简可得:

$$r(h) = \sqrt{\frac{\max(10, \pi\left(\frac{0.596h-0.3}{2}\right)^2)}{\pi}} \quad 2.5 \leq r \leq \min(\frac{L}{2}, \frac{W}{2}) \quad 1 \leq h \leq 10$$

求解该函数满足条件的最小值,我们用图像法进行求解,得到如图3:

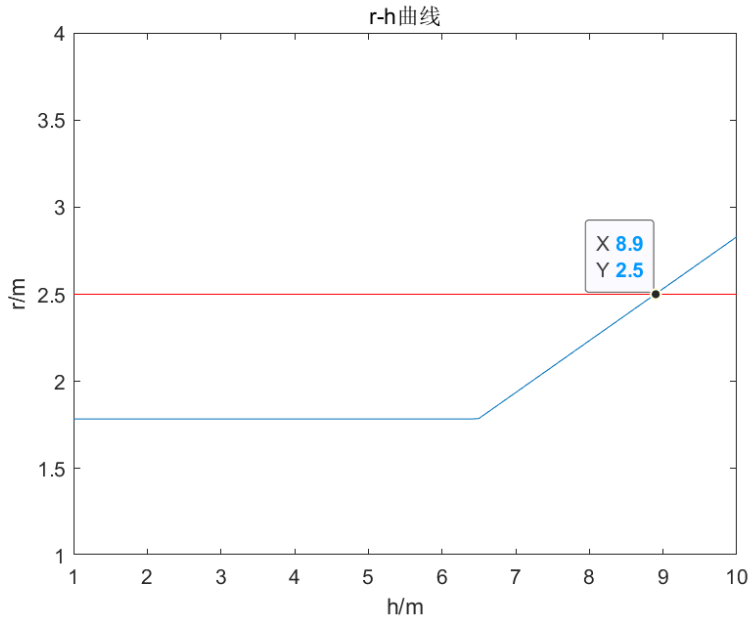

图 3: 占地面积半径与树高关系

得到最小半径是 2.5。

- 对于 $500m \times 500m$ 的土地，求解种植树木数量的最大，抽象问题成为“把圆形放入矩形的 packing”问题。

  圆形放入矩形的 packing 问题 [1-2]:

  - Packing 问题，是一类组合优化问题，研究的是把一组较小的图形相互间无嵌入的放入较大的图形，其最终目标是寻求最优（放入最多的小图形）的放置方式。有效地求解此类问题，可以较好地利用资源、减少浪费，这对实际的生产领域，如物体的运输、堆放以及原料的下料等领域，具有不可估量的经济效益。
  - 圆形放入矩形的 packing 问题：就是将一组圆形相互无嵌入放入一个矩形之中，最终目标是寻求放入最多的小圆形的放置数量和放置方式。
  - Packing 问题通常都是 NP 难度的，至今无严格有效的方法能保证在合理的计算时间内找出 NP 难问题的最优解。受自然界和人类社会中智慧的启发，人们提出了有

效地近似求解具有难度的圆形问题的启发式方法。本题同样也是基于这种启发式方法对问题进行求解。

- 本题采用启发式算法之中的" 拟物算法" 进行快速的近似求解。
- 算法思想:

1. 算法实现是将圆想象成光滑的圆饼,将圆饼放入容器内,若容器的圆饼有挤压,就会由于挤压弹性力的作用产生一系列运动,作为运动的结果,可能就存在问题解的确立——物体间两两互不嵌入。
2. 首先,给出 n 个小圆,尝试将它们放入大的矩形当中。若成功,则继续尝试将 n+1 个小圆放入其中;否则,算法结束,n 就是最优解。下面详细叙述怎样进行有效地尝试以期能够把这 n 个小圆放入到大矩形当中。
3. 其中 R 为小圆的半径长,x_i、y_i 分别是第 i 个小圆的横纵坐标,L 为大矩形的长,W 为大矩形的宽。(下同)
4. 为寻找满足条件的解,先随机的把 n 个圆的圆心放入到大矩形当中。
5. 然后定义与圆形间以及圆形与边界间的嵌入深度相关的**"势能"**。
   - 第 i 个与第 j 个圆之间的势能:

$$d_{i,j} = \begin{cases} 2R - \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} & , \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} < 2R \\ 0 & , else \end{cases}$$

   - 第 i 个圆与矩形的四条边之间的势能:

$$d_{i,n+1} = \begin{cases} R - y_i & , R - y_i > 0 \\ 0 & , else \end{cases}$$

$$d_{i,n+2} = \begin{cases} R - x_i & , R - x_i > 0 \\ 0 & , else \end{cases}$$

$$d_{i,n+3} = \begin{cases} y_i + R - W & , y_i + R - W > 0 \\ 0 & , else \end{cases}$$

$$d_{i,n+1} = \begin{cases} x_i + R - L & , x_i + R - L > 0 \\ 0 & , else \end{cases}$$

- 最后,定义整个系统的势能值:

$$U(X) = \sum_{i=1}^{n} \sum_{j=1}^{n+4} d_{i,j}^2 \quad , where X = (x_1, \cdots, x_n, y_1, \cdots, y_n)$$

- 很容易理解,寻找放置 n 个圆的过程,即是寻找 $U(X)$ 的最小值是否为 0 的过程。

6. 不难看出,$U(X)$ 是非连续函数,想要去求解它的最小值只能用搜索算法,例如模拟退火算法,蚁群算法,遗传算法,或者模式搜索算法等。其中拟物算法在求解等圆 packing 问题时,存在计算后期的收敛速度相当慢、盲目搜索等问题。针对拟物算法的不足,提出了以下几点改进策略:通过烧荒策略调整圆饼的移动次序,充分利用圆饼每次移动的信息,加快了算法的演化速度;通过横向搜索策略解决在全局最小值附近可能出现势能下降停滞不前的问题;本题中拟物算法当圆饼数量小于 60 左右时求解该函数非常精确,因此我们对该算法进行两个优化。

- **第一个优化**

我们将 $U(X)$ 这个非连续函数进行平滑处理，使其成为连续函数，采用 $Sigmoid$ 函数将 $d_{i,j} = \max(0, a)$ 转化成为 $d_{i,j} = \frac{a}{1+e^{-ab}}$，$b$ 取一个较大值，在这里我们选取 $b = 10000$，当 $a > 0$，$d_{i,j}$ 无限趋向于 a，当 $a < 0$ 时，$d_{i,j}$ 无限趋向于 0，这样做之后我们可以用其他算法求解该函数的最小值，并且速度得到了一个数量级甚至两个数量级的提升。

- **第二个优化**

  此问题树木的数量远远超过了 60，也就是圆饼的数量远远超过了 60，求解复杂度非常高，因此我们将 $500m \times 500m$ 的土地分成若干个 $25m \times 25m$ 的土地，对每一块土地进行求解，这样做既保证了拟物算法求解的精确性，又非常显著的降低了求解的复杂度，使其能够在较短的时间内求解出来。

- 得到如图4种植结果：**最多可以种植** 10000 **颗树**。

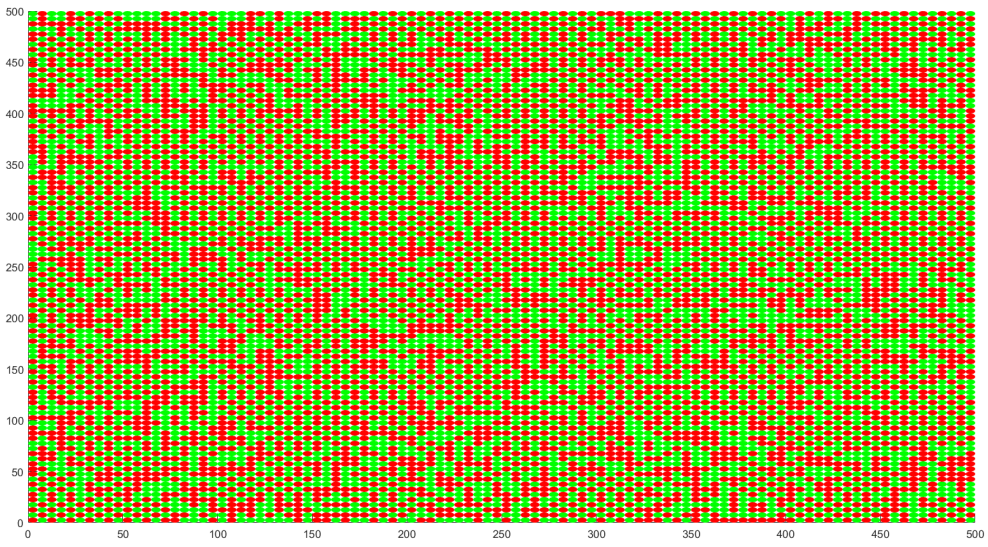

图 4: 树木种植图 (红绿均代表一棵树)

5.2 问题二的模型建立与求解

求解目标和第一问一样,最优解方案的存在与否取决于树木所占面积抽象成的“圆饼”间“势能”能否为 0。

总的势能可以划分为三个部分:原有树木抽象为的“圆饼”之间的势能、需要新栽种的树木抽象成的“圆饼”之间的势能、原有树木抽象和新产生的树木抽象为的“圆饼”间的势能。

矩形土地上原有树木抽象成的“圆饼”之间势能已经为 0,要使得最终势能为 0,只需验证原有树木和新产生的树木抽象为的“圆饼”间的势能是否为 0。

假设现存树的坐标为:

$$Y = (u_1, \cdots, u_m, v_1, \cdots, v_m)$$

原有树木和新产生的树木抽象为的“圆饼”间的势能:

$$g_{i,j} = \begin{cases} 2R - \sqrt{(x_i - u_j)^2 + (y_i - v_j)^2} & , \sqrt{(x_i - u_j)^2 + (y_i - v_j)^2} < 2R \\ 0 & , else \end{cases}$$

整个系统的总势能函数为:

$$U(X,Y) = \sum_{i=1}^{n} \sum_{j=1}^{n+4} d_{i,j}^2 + \sum_{i=1}^{n} \sum_{j=1}^{m} g_{i,j}^2$$

$$where X = (x_1, \cdots, x_n, y_1, \cdots, y_n) \quad Y = (u_1, \cdots, u_m, v_1, \cdots, v_m)$$

随机生成一些现存树木的位置,得到还可以种植的图如下:

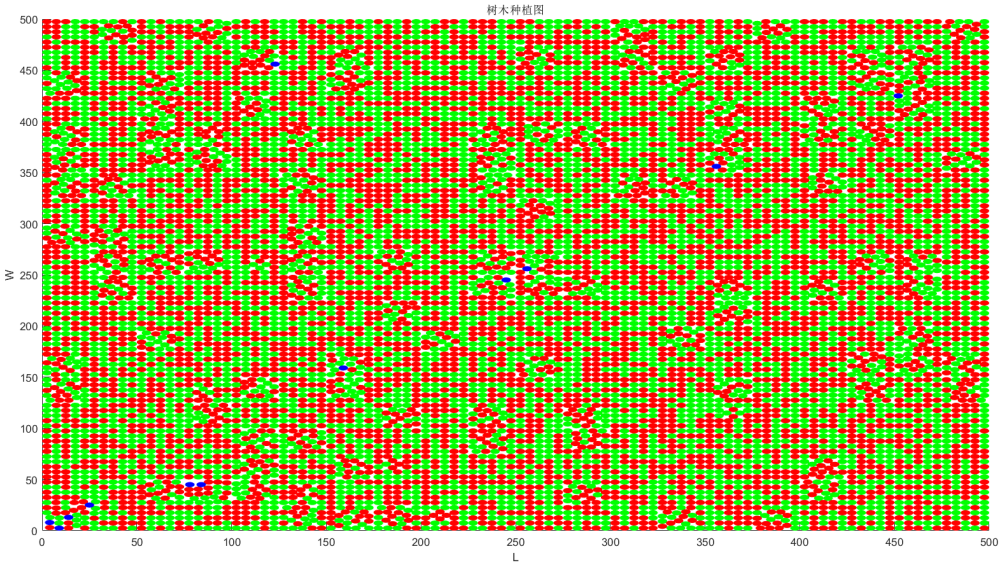

图 5: 树木种植图 (蓝色代表已经存在的树)

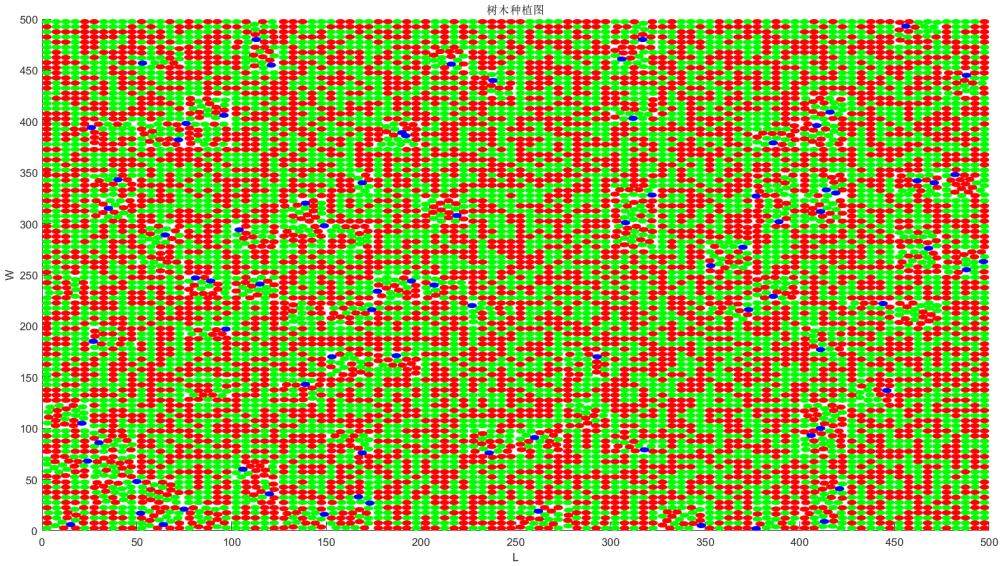

图 6: 树木种植图 (蓝色代表已经存在的树)

5.3 问题三的模型建立与求解

在 $500m500m$ 的土地上已经种植了一些树木，每棵树木的高度一致，需要调整它们的高度，以最大化覆盖面积。

我们可以将问题建模为一个优化问题，其中目标是最大化树的高度，约束条件是每对树之间的距离不小于它们的占地面积的一半。

具体来说，假设有 n 棵树，它们的坐标分别为 $(x_1, y_1), \ldots, (x_n, y_n)$，我们需要求出树的高度 h，使得它们的冠幅最大，即 $\max h$ 最大。

根据题目要求，我们需要满足以下约束条件：

对于树的占地半径，由第一问可知，占地面积半径 r 和树的高度 h 有以下关系：

$$r(h) = \sqrt{\frac{\max(10, \pi\left(\frac{0.596h-0.3}{2}\right)^2)}{\pi}} \quad 2.5 \le r \le \min(\frac{L}{2}, \frac{W}{2}) \quad 1 \le h \le 10$$

对于任意 $i, j \in 1, \ldots, n,\ i \neq j$，两棵树之间的距离 d_{ij} 不小于它们的占地半径，即 $d_{ij} \ge r + r$。

注意到 d_{ij} 可以通过坐标之间的欧几里得距离计算得到：

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

因此，我们可以将上述约束条件写成以下形式：

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \ge 2\sqrt{\frac{\max(10, \pi\left(\frac{0.596h-0.3}{2}\right)^2)}{\pi}}$$

$$2.5 \le r \le \min(\frac{L}{2}, \frac{W}{2}) \quad 1 \le h \le 10, \quad i, j \in 1, \ldots, n, \ i \ne j.$$

随机生成一些现存树木的位置，得到当树的高度是 $9.9996m$ 时，有最大的覆盖面积 $471203.9339m^2$,其种植图如图7:

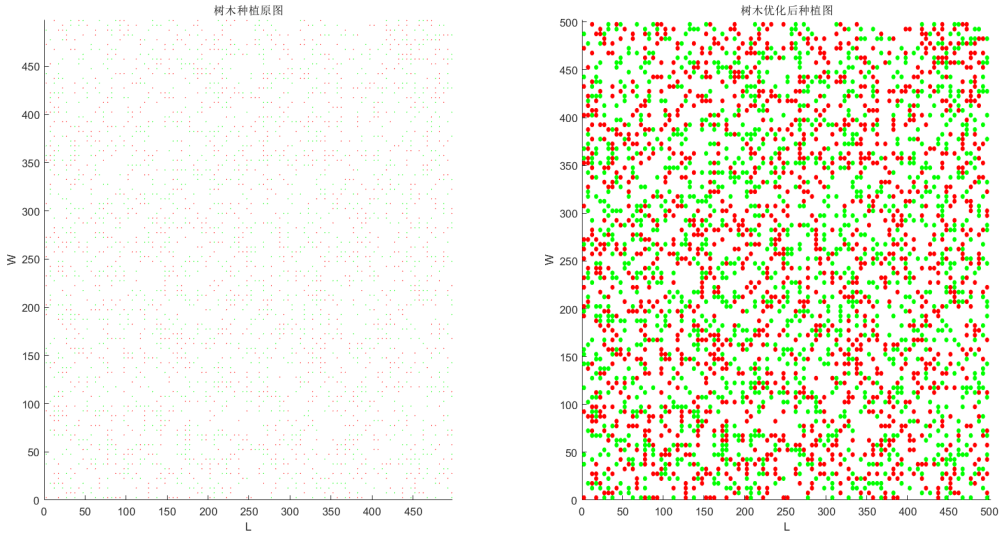

图 7: 种植前后对比图

5.4 问题四的模型建立与求解

要使种植数量最多,根据第一问,树木的占地面积半径 r 依旧是最小值 2.5。

设置 LW 为变量。

增加约束条件 $L \times W = 300$ 。

依次遍历 L 的边长,根据约束条件 $L \times W = 300$ 求得 W ,运用第一问的拟物算法进行求解,找出种植的最大数量和最优方案。

得到如图8种植结果:**当 $L = 60, W = 5$ 时,最多可以种植 12 棵树**。

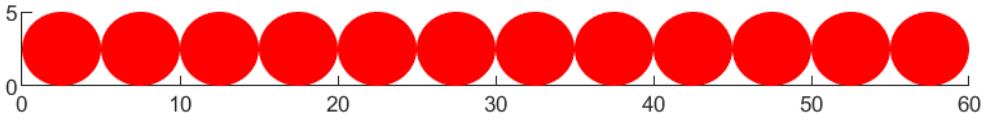

图 8: 种植图

6 模型分析与改进

6.1 模型优点

1. 本文基于 Packing 模型及其优化，采用启发式的方法进行问题的最优化，并且在给定的假设下，运用 Packing 算法求解得到的结果与实际生活较为符合。
2. Packing 模型主要的运用场景是离散性数据，但是在实际生活中，本类问题求解是连续性问题，所以本文的优化采用 Smooth 函数，将离散的数据连续化，更加符合实际求解情况。
3. Packing 模型的求解数量限制在 100 以内，所以本文中将土地合理划分，然后加上合理的限制最优化模型。
4. 由于土地划分，所以对于一棵树木的归属问题，本文采用势能求解的方法合理解决。

6.2 模型缺点

1. 本文在处理树高问题，为了方便问题求解和问题优化，简化为树高一致。
2. 本文由于问题的求解是在树木高度一致的前提下完成的，因此在一些对土地利用率有要求的题目中该模型的适用性不高。
3. 本文在解题中半径是基于题干求出的具体值，因此在模型的泛化能力上会有所欠缺。

6.3 模型改进与推广

本文使用的土地划分优化，可以将大规模问题划分为小问题求解，在实际生活中具有较大意义，能够更好的解决实际生活中大规模的城市绿化优化问题。

然而在实际生活中，面对大规模的优化问题，其计算复杂度显著上升，所以导致基于本方法的计算时间开销巨大，故后续若基于本方法求解，应提高其计算速度，降低算法复杂度，降低时间开销。

参考文献

[1] 康雁. 求解圆形 packing 问题的启发式方法. 华中科技大学计算机学院, 中国科学院研究生院博士学位论文, 2002.

[2] 盛建. 求解等圆 Packing 问题的拟物拟人算法改进策略研究. 华中科技大学硕士学位论文, 2009.

附录一

支撑文件清单		
文件夹名	**文件名**	**含义**
数据	data_1	问题二数据
	data_3	问题三数据
代码	code1.1.m	冠幅与树高的拟合曲线
	code1.2.m	问题一拟合残差图
	code1.3.m	问题一 r-h 曲线图
	code1.4.m	问题一树木种植图
	code2.1.m	问题二图像一
	code2.2.m	问题二图像二
	code3.m	问题三代码
	code4.m	问题四代码

附录二

A 问题一代码

A.1 冠幅与树高的拟合曲线

```
clc; clear ;
% 拟合高度 (h) 与冠幅(p)
% 拟合函数 p = a + bh
h = [5, 10, 15, 20, 25]'; % 高度(m)
p = [2.8, 5.5, 8.5, 11.9, 14.5]'; % 冠幅(m)

X = [ones(size(h)) h] ;
[b,bint,r,rint,stats] = regress(p,X) ;

hi = 0:0.1:25; % 拟合区间, 这里设置为0到25, 步长为0.1
pi = [ones(size(hi')) hi'] * b ;

figure ;
plot(h,p,'+',hi,pi); % 绘制原始数据散点图和拟合曲线
legend('原始数据','拟合数据');
title('拟合下的h-p曲线');
xlabel('h/m'),ylabel('p/m') ;
```

A.2 拟合残差图

```
clc; clear ;
% 拟合高度 (h) 与冠幅(p)
% 拟合函数 p = a + bh
h = [5, 10, 15, 20, 25]'; % 高度(m)
p = [2.8, 5.5, 8.5, 11.9, 14.5]'; % 冠幅(m)

X = [ones(size(h)) h] ;
[b,bint,r,rint,stats] = regress(p,X) ;

figure ;
```

```
rcoplot(r,rint) ; % 残差图
```

A.3 r-h 曲线图

```
%寻找最小树木占地半径的图像代码, 求出的最小半径是2.5
clc ; clear all ;
L = 500 ;
W = 500 ;
figure ;
h = 1:0.1:10 ;
r = fun(h) ;
line = repmat(2.5,size(h,2), 1) ;
plt = plot(h,r, h, line, 'r') ;
axis([1 10 1 min(min(4,W/2),L/2)]) ;
xlabel("h/m") ;
ylabel("r/m") ;
title("r-h曲线") ;
set(plt(2),'Color',[1 0 0]);
datatip(plt(2),'DataIndex',80,'Location','northwest');

function r = fun(h)
    p = 0.596*h-0.3 ;
    A = pi * (p / 2).^2 ;
    S = max(10, A) ;
    r = sqrt(S / pi) ;
end
```

A.4 树木种植图

```
clc ; clear all ;

e1 = 0.5; % 精度, 小于该值都认为函数的最小值是0,因为平滑函数的缘故, 需要将精度调高一
    些, 经过调试, 精度在0.5左右比较好

W = 500 ; % 矩形土地的长
L = 500 ; % 矩形土地的宽
R = 2.5 ; % 树的半径
```

```
W_block = 25 ;  % 每一块的W
L_block = 25 ;  % 每一块的L

final_x = [];
final_y = [];
final_n = 0;

n_pre_max = floor(L_block / (2 * R)) * floor(W_block / (2 * R)) ;
for y_move = 1:W/W_block
    for x_move = 1:L/L_block
        %圆平移
        %%%%%%%%%%每张图的初始化%%%%%%%%%%%%%%%%%%%%%%
        n_left = max(0, floor(L_block / (2 * R)) * floor(W_block / (2 * R))-100) ; % 圆
            形的最小数量（一定可以）
        n_right = ceil(L_block * W_block / (pi * R * R)) ;  % 圆形的最大数量（不一定可
            以）
        n_max = floor(L_block / (2 * R)) * floor(W_block / (2 * R)) ;  % 二分答案的最大
            圆的数量
        x_and_y_position = [] ; % 最大数量下的各个点的坐标

        % 初始化，圆形的最小数量，均匀排布。
        x = rand(floor(L_block / (2 * R)) * floor(W_block / (2 * R)), 1) * L_block ;
        y = rand(floor(L_block / (2 * R)) * floor(W_block / (2 * R)), 1) * W_block ;
        cnt = 1 ; m = floor(L_block / (2 * R)) ;
        for i = 1:floor(L_block / (2 * R))
            for j = 1:floor(W_block / (2 * R))
                x(cnt) = R + (j - 1) * (2 * R) ;
                y(cnt) = R + (i - 1) * (2 * R) ;
                cnt = cnt + 1 ;
            end
        end
        x_and_y_position = [x ; y] ;

        % 因为圆形的数量 满足单调性，运用二分算法求解满足特定土地的最大圆形数量
        n_max_esp = 100 ;
        %%%%%%%%%%每张图的初始化%%%%%%%%%%%%%%%%%%%%%%
        Iter = 20 ;  % 每一块的迭代次数
        cnt = 0 ;
```

```
        while (cnt < Iter && n_pre_max > n_max) || cnt == 0
            cnt = cnt + 1 ;
            left = max(n_max, n_left) ;
            right = n_right ;
            while left <= right
                mid = floor((left + right) / 2) ;
                disp("现在正在计算: " + num2str(mid)) ;
                [result, solution] = check(mid, L_block, W_block, R, e1) ;
                if result == 1
                    if n_max < mid
                        n_max = mid ;

                        x_and_y_position = solution ;
                        n_max_esp = result ;
                    end
                    if n_max == mid && n_max_esp > result
                        x_and_y_position = solution ;
                        n_max_esp = result ;
                    end
                    left = mid + 1 ;
                    disp(num2str(mid) + "个圆形可以容纳下!") ;
                else
                    right = mid - 1 ;
                    disp(num2str(mid) + "个圆形不能容纳下!") ;
                end
            end
        end
        n_pre_max = max(n_pre_max, n_max) ;
        final_x = cat(1, final_x, x_and_y_position(1:n_max) + (x_move - 1) * L_block);
        final_y = cat(1, final_y, x_and_y_position(n_max+1:end) + (y_move - 1) *
            W_block);
        final_n = final_n + n_max;
    end

end

% 绘制圆形
x = final_x;
y = final_y;
```

```
figure ;
for i = 1:final_n
    if mod(i ,2) == 1
        rectangle('Position', [x(i)-R, y(i)-R, 2*R, 2*R], 'Curvature', [1, 1], '
            FaceColor', 'r', 'EdgeColor', 'none');
    else
        rectangle('Position', [x(i)-R, y(i)-R, 2*R, 2*R], 'Curvature', [1, 1], '
            FaceColor', 'g', 'EdgeColor', 'none');
    end
end
axis([0 L 0 W]);
xlabel("L") ;
ylabel("W") ;
title("树木种植图")
disp("最多可以种植: "+final_n+"树") ;

% 二分函数的check函数
function [result, solution] = check(m, L_block, W_block, R, e1)

    % 初始化迭代点
    x = rand(m, 1) * L_block ;
    y = rand(m, 1) * W_block ;
    X = [x ; y] ;

    % 约束变量的上界和下界
    lb = [repmat(R, m, 1) ; repmat(R, m, 1)] ;
    ub = [repmat(L_block-R, m, 1) ; repmat(W_block-R, m, 1)] ;

    % 将固定参数传递给 objfun
    objfun = @(X)objectiveFcn(X,L_block,W_block,R) ;

    % 运用fmincon函数求解函数最小值
    %options = optimoptions('ga','PopulationSize', 50, 'Generations', 100, "HybridFcn
        ","fmincon");
    %[solution,objectiveValue] = ga(objfun,2*m,[],[],[],[],lb,ub,[],options); % 遗传算
        法
    %[solution,objectiveValue] = particleswarm(objfun,2*m,lb,ub); % 多粒度算法
    %[solution,objectiveValue] = patternsearch(objfun,X,[],[],[],[],lb,ub) ; % 模式搜索
    options = optimoptions("fmincon","MaxFunctionEvaluations",10000);
```

```
    [solution,objectiveValue] = fmincon(objfun,X,[],[],[],[],lb,ub,[],options);


    % 返回check的结果, 表示
    result = objectiveValue < e1 ;
end

% 目标函数, 采用拟物算法,参考"求解圆形packing问题的启发式方法_康雁"
function f = objectiveFcn(X, L_block, W_block, R)
    n = size(X, 1) / 2 ;
    x = X(1:n) ;
    y = X(n+1:end) ;
    d = zeros(n, n+4) ;
    for i = 1:n
        for j = 1:n
            if i == j
                continue ;
            end
            tp = sqrt((x(i)-x(j))^2 + (y(i)-y(j))^2) ;
            d(i,j) = smooth(2*R - tp) ;
        end
        d(i, n+1) = smooth(R-y(i)) ;
        d(i, n+2) = smooth(R-x(i)) ;
        d(i, n+3) = smooth(R-W_block+y(i)) ;
        d(i, n+4) = smooth(R-L_block+x(i)) ;
    end
    %disp(d) ;
    f = sum(sum(d.^2)) ;
end

% 对max(0,a)进行平滑处理, 非连续函数运算较慢, 我们对非连续函数进行平滑处理, 使其成为连
    续函数, 在求解答案时获得了一个数量级甚至两个数量级的提升, 与此带来的害处是精度降
    低, 需要调节精度, 不过这是非常值得的。
% 平滑函数是 Sigmoid 函数
function f = smooth(a)
    f = a ./ (1 + exp(-a*10000)) ;
end
```

B 问题二代码

B.1 问题一图像一

```
clc ; clear all ;

e1 = 0.5;



W = 500 ; % 矩形土地的长
L = 500 ; % 矩形土地的宽
R = 2.5 ;  % 树的半径

% 第一幅图数据
x_exist = [4 ; 9 ; 14 ; 25 ; 123 ; 78 ; 84 ; 159 ; 245 ; 256 ; 356 ; 452] ;  % 已经存在
    的树的x坐标
y_exist = [8 ; 2.5 ; 13 ; 25 ; 456 ; 45 ; 45 ; 159 ; 245 ; 256 ; 356 ; 425] ;  % 已经存
    在的树的y坐标



W_block = 25 ;  % 每一块的W
L_block = 25 ;  % 每一块的L

final_x = [];
final_y = [];
final_n = 0;
n_pre_max = floor(L_block / (2 * R)) * floor(W_block / (2 * R)) ;
for y_move = 1:W/W_block
    for x_move = 1:L/L_block
        %圆平移
        %%%%%%%%%%每张图的初始化%%%%%%%%%%%%%%%%%%%%%%
        n_left = 0 ; % 圆形的最小数量 (一定可以)
        n_right = ceil(L_block * W_block / (pi * R * R)) + 1;  % 圆形的最大数量 (不一定
            可以)
        n_max = 0 ;  % 二分答案的最大圆的数量
        x_and_y_position = [] ; % 最大数量下的各个点的坐标

        Y = [x_exist - (x_move - 1) * L_block;  y_exist - (y_move - 1) * W_block] ;
```

```
% 因为圆形的数量 满足单调性，运用二分算法求解满足特定土地的最大圆形数量
n_max_esp = 100 ;
Iter = 20 ;  % 每一块的迭代次数
cnt = 0 ;
while (cnt < Iter && n_pre_max > n_max) || cnt == 0
    cnt = cnt + 1 ;
    left = max(n_max, n_left) + 1 ;
    right = n_right ;
    while left < right
        mid = floor((left + right) / 2) ;
        if mid == 0
            break ;
        end
        disp("现在正在计算: " + num2str(mid)) ;
        [result, solution] = check(mid, L_block, W_block, R, e1, Y) ;
        if result == 1
            if n_max < mid
                n_max = mid ;
                x_and_y_position = solution ;
                n_max_esp = result ;
            end
            if n_max == mid && n_max_esp > result
                x_and_y_position = solution ;
                n_max_esp = result ;
            end
            left = mid + 1 ;
            disp(num2str(mid) + "个圆形可以容纳下!") ;
        else
            right = mid - 1 ;
            disp(num2str(mid) + "个圆形不能容纳下!") ;
        end
    end
end
if n_max == 0
    continue ;
end
n_pre_max = max(n_pre_max, n_max) ;
final_x = cat(1, final_x, x_and_y_position(1:n_max) + (x_move - 1) * L_block);
final_y = cat(1, final_y, x_and_y_position(n_max+1:end) + (y_move - 1) *
```

```
            W_block);
        final_n = final_n + n_max;
    end

end



% 绘制圆形
x = final_x;
y = final_y;
figure ;
for i = 1:final_n
    if mod(i ,2) == 1
        rectangle('Position', [x(i)-R, y(i)-R, 2*R, 2*R], 'Curvature', [1, 1], '
            FaceColor', 'r', 'EdgeColor', 'none');
    else
        rectangle('Position', [x(i)-R, y(i)-R, 2*R, 2*R], 'Curvature', [1, 1], '
            FaceColor', 'g', 'EdgeColor', 'none');
    end
end
for i = 1:size(x_exist, 1)
    rectangle('Position', [x_exist(i)-R, y_exist(i)-R, 2*R, 2*R], 'Curvature', [1, 1],
        'FaceColor', 'b', 'EdgeColor', 'none');
end
axis([0 L 0 W]);
xlabel("L") ;
ylabel("W") ;
title("树木种植图")

disp("最多还可以种植: "+final_n+"树") ;

% 二分函数的check函数
function [result, solution] = check(m, L_block, W_block, R, e1, Y)

    % 初始化迭代点
    x = rand(m, 1) * L_block ;
    y = rand(m, 1) * W_block ;
    X = [x ; y] ;
```

```
    % 约束变量的上界和下界
    lb = [repmat(R, m, 1) ; repmat(R, m, 1)] ;
    ub = [repmat(L_block-R, m, 1) ; repmat(W_block-R, m, 1)] ;

    % 将固定参数传递给 objfun
    objfun = @(X)objectiveFcn(X,L_block,W_block,R,Y) ;

    % 运用fmincon函数求解函数最小值
    %options = optimoptions('ga','PopulationSize', 50, 'Generations', 100, "HybridFcn
        ","fmincon");
    %[solution,objectiveValue] = ga(objfun,2*m,[],[],[],[],lb,ub,[],options); % 遗传算
        法
    %[solution,objectiveValue] = particleswarm(objfun,2*m,lb,ub); % 多粒度算法
    %[solution,objectiveValue] = patternsearch(objfun,X,[],[],[],[],lb,ub) ; % 模式搜索
    options = optimoptions("fmincon","MaxFunctionEvaluations",10000);
    [solution,objectiveValue] = fmincon(objfun,X,[],[],[],[],lb,ub,[],options);

    % 返回check的结果, 表示
    result = objectiveValue < e1 ;
end

% 目标函数, 采用拟物算法,参考"求解圆形packing问题的启发式方法_康雁"
function f = objectiveFcn(X, L_block, W_block, R, Y)
    n = size(X, 1) / 2 ;
    x = X(1:n) ;
    y = X(n+1:end) ;
    d = zeros(n, n+4) ;
    for i = 1:n
        for j = 1:n
            if i == j
                continue ;
            end
            tp = sqrt((x(i)-x(j))^2 + (y(i)-y(j))^2) ;
            d(i,j) = smooth(2*R - tp) ;
        end
        d(i, n+1) = smooth(R-y(i)) ;
        d(i, n+2) = smooth(R-x(i)) ;
        d(i, n+3) = smooth(R-W_block+y(i)) ;
        d(i, n+4) = smooth(R-L_block+x(i)) ;
```

```
    end

    m = size(Y, 1) / 2 ;
    x1 = Y(1:m) ;
    y1 = Y(m+1:end) ;
    g = rand(n, m) ;
    for i = 1:n
        for j = 1:m
            tp = sqrt((x(i)-x1(j))^2 + (y(i)-y1(j))^2) ;
            g(i, j) = smooth(2*R - tp) ;

        end
    end

    %disp(d) ;
    f = sum(sum(d.^2)) + sum(sum(g.^2)) ;
end

% 对max(0,a)进行平滑处理，非连续函数运算较慢，我们对非连续函数进行平滑处理，使其成为连续函数，在求解答案时获得了一个数量级甚至两个数量级的提升，与此带来的害处是精度降低，需要调节精度，不过这是非常值得的。
% 平滑函数是 Sigmoid 函数
function f = smooth(a)
    f = a ./ (1 + exp(-a*10000)) ;
end
```

B.2 问题二图像二

```
clc ; clear all ;

e1 = 0.5;


W = 500 ; % 矩形土地的长
L = 500 ; % 矩形土地的宽
R = 2.5 ;  % 树的半径

% 第二幅图
```

```
% 已经存在的树的x坐标
x_exist = [488; 89; 167; 120; 153; 293; 177; 207; 35; 308; 406; 468; 139; 53; 50; 81;
    15; 227; 40; 52; 174; 195; 377; 173; 24; 411; 497; 353; 106; 471; 421; 169; 317;
    377; 306; 113; 27; 370; 76; 72; 262; 389; 97; 190; 482; 149; 238; 312; 75; 216;
    416; 322; 64; 386; 462; 488; 373; 192; 187; 456; 219; 318; 386; 65; 444; 236; 413;
    115; 409; 446; 121; 260; 419; 169; 21; 149; 414; 411; 348; 96; 411; 139; 104; 30;
    26];
% 已经存在的树的y坐标
y_exist = [255; 244; 33; 36; 170; 170; 234; 240; 315; 301; 93; 276; 143; 457; 48; 247;
    6; 220; 343; 17; 216; 244; 327; 27; 68; 177; 263; 259; 60; 340; 41; 340; 480; 2;
    461; 480; 185; 277; 398; 382; 19; 302; 197; 389; 348; 298; 440; 403; 21; 456; 409;
    328; 6; 229; 342; 445; 216; 386; 171; 493; 308; 79; 379; 289; 222; 76; 9; 241; 396;
     137; 455; 91; 330; 76; 105; 16; 333; 312; 5; 406; 100; 320; 294; 86; 394];


W_block = 25 ;  % 每一块的W
L_block = 25 ;  % 每一块的L

final_x = [];
final_y = [];
final_n = 0;
n_pre_max = floor(L_block / (2 * R)) * floor(W_block / (2 * R)) ;
for y_move = 1:W/W_block
    for x_move = 1:L/L_block
        %圆平移
        %%%%%%%%%%每张图的初始化%%%%%%%%%%%%%%%%%%%%%%
        n_left = 0 ; % 圆形的最小数量（一定可以）
        n_right = ceil(L_block * W_block / (pi * R * R)) + 1;  % 圆形的最大数量（不一定
            可以）
        n_max = 0 ;  % 二分答案的最大圆的数量
        x_and_y_position = [] ; % 最大数量下的各个点的坐标

        Y = [x_exist - (x_move - 1) * L_block;  y_exist - (y_move - 1) * W_block] ;

        % 因为圆形的数量 满足单调性，运用二分算法求解满足特定土地的最大圆形数量
        n_max_esp = 100 ;
        Iter = 20 ;  % 每一块的迭代次数
        cnt = 0 ;
```

```
        while (cnt < Iter && n_pre_max > n_max) || cnt == 0
            cnt = cnt + 1 ;
            left = max(n_max, n_left) + 1 ;
            right = n_right ;
            while left < right
                mid = floor((left + right) / 2) ;
                if mid == 0
                    break ;
                end
                disp("现在正在计算: " + num2str(mid)) ;
                [result, solution] = check(mid, L_block, W_block, R, e1, Y) ;
                if result == 1
                    if n_max < mid
                        n_max = mid ;
                        x_and_y_position = solution ;
                        n_max_esp = result ;
                    end
                    if n_max == mid && n_max_esp > result
                        x_and_y_position = solution ;
                        n_max_esp = result ;
                    end
                    left = mid + 1 ;
                    disp(num2str(mid) + "个圆形可以容纳下!") ;
                else
                    right = mid - 1 ;
                    disp(num2str(mid) + "个圆形不能容纳下!") ;
                end
            end
        end
        if n_max == 0
            continue ;
        end
        n_pre_max = max(n_pre_max, n_max) ;
        final_x = cat(1, final_x, x_and_y_position(1:n_max) + (x_move - 1) * L_block);
        final_y = cat(1, final_y, x_and_y_position(n_max+1:end) + (y_move - 1) *
            W_block);
        final_n = final_n + n_max;
    end
```

```
end



% 绘制圆形
x = final_x;
y = final_y;
figure ;
for i = 1:final_n
    if mod(i ,2) == 1
        rectangle('Position', [x(i)-R, y(i)-R, 2*R, 2*R], 'Curvature', [1, 1], '
            FaceColor', 'r', 'EdgeColor', 'none');
    else
        rectangle('Position', [x(i)-R, y(i)-R, 2*R, 2*R], 'Curvature', [1, 1], '
            FaceColor', 'g', 'EdgeColor', 'none');
    end
end
for i = 1:size(x_exist, 1)
    rectangle('Position', [x_exist(i)-R, y_exist(i)-R, 2*R, 2*R], 'Curvature', [1, 1],
        'FaceColor', 'b', 'EdgeColor', 'none');
end
axis([0 L 0 W]);
xlabel("L") ;
ylabel("W") ;
title("树木种植图")

disp("最多还可以种植: "+final_n+"树") ;




% 二分函数的check函数
function [result, solution] = check(m, L_block, W_block, R, e1, Y)

    % 初始化迭代点
    x = rand(m, 1) * L_block ;
    y = rand(m, 1) * W_block ;
    X = [x ; y] ;

    % 约束变量的上界和下界
```

```
    lb = [repmat(R, m, 1) ; repmat(R, m, 1)] ;
    ub = [repmat(L_block-R, m, 1) ; repmat(W_block-R, m, 1)] ;

    % 将固定参数传递给 objfun
    objfun = @(X)objectiveFcn(X,L_block,W_block,R,Y) ;

    % 运用fmincon函数求解函数最小值
    %options = optimoptions('ga','PopulationSize', 50, 'Generations', 100, "HybridFcn
        ","fmincon");
    %[solution,objectiveValue] = ga(objfun,2*m,[],[],[],[],lb,ub,[],options); % 遗传算
        法
    %[solution,objectiveValue] = particleswarm(objfun,2*m,lb,ub); % 多粒度算法
    %[solution,objectiveValue] = patternsearch(objfun,X,[],[],[],[],lb,ub) ; % 模式搜索
    options = optimoptions("fmincon","MaxFunctionEvaluations",10000);
    [solution,objectiveValue] = fmincon(objfun,X,[],[],[],[],lb,ub,[],options);

    % 返回check的结果, 表示
    result = objectiveValue < e1 ;
end

% 目标函数, 采用拟物算法,参考"求解圆形packing问题的启发式方法_康雁"
function f = objectiveFcn(X, L_block, W_block, R, Y)
    n = size(X, 1) / 2 ;
    x = X(1:n) ;
    y = X(n+1:end) ;
    d = zeros(n, n+4) ;
    for i = 1:n
        for j = 1:n
            if i == j
                continue ;
            end
            tp = sqrt((x(i)-x(j))^2 + (y(i)-y(j))^2) ;
            d(i,j) = smooth(2*R - tp) ;
        end
        d(i, n+1) = smooth(R-y(i)) ;
        d(i, n+2) = smooth(R-x(i)) ;
        d(i, n+3) = smooth(R-W_block+y(i)) ;
        d(i, n+4) = smooth(R-L_block+x(i)) ;
    end
```

```
    m = size(Y, 1) / 2 ;
    x1 = Y(1:m) ;
    y1 = Y(m+1:end) ;
    g = rand(n, m) ;
    for i = 1:n
        for j = 1:m
            tp = sqrt((x(i)-x1(j))^2 + (y(i)-y1(j))^2) ;
            g(i, j) = smooth(2*R - tp) ;

        end
    end

    %disp(d) ;
    f = sum(sum(d.^2)) + sum(sum(g.^2)) ;
end

% 对max(0,a)进行平滑处理，非连续函数运算较慢，我们对非连续函数进行平滑处理，使其成为连续函数，在求解答案时获得了一个数量级甚至两个数量级的提升，与此带来的害处是精度降低，需要调节精度，不过这是非常值得的。
% 平滑函数是 Sigmoid 函数
function f = smooth(a)
    f = a ./ (1 + exp(-a*10000)) ;
end
```

C 问题三代码

```
perm = randperm(length(final_x));
x = final_x(perm(1:5000));
y = final_y(perm(1:5000));
n = 5000;
dis = 100000000;


subplot(1,2,1);
R = 0.5; % set R to 0.5
L = max(x) + 2*R;
W = max(y) + 2*R;
```

```
for i = 1:n
    if mod(i, 2) == 1
       rectangle('Position', [x(i)-R, y(i)-R, 2*R, 2*R], 'Curvature', [1, 1], '
           FaceColor', 'r', 'EdgeColor', 'none');
    else
        rectangle('Position', [x(i)-R, y(i)-R, 2*R, 2*R], 'Curvature', [1, 1], '
            FaceColor', 'g', 'EdgeColor', 'none');
    end
end
axis([0 L 0 W]);
xlabel("L") ;
ylabel("W") ;
title("树木种植原图");

% rest of the code remains the same

for i = 1:n
    for j = 1:n
        if(i == j)
            continue
        end
        if(sqrt((x(i) - x(j)) ^ 2 + (y(i) - y(j)) ^ 2 ) > 7.5)
            dis = min(dis, sqrt((x(i) - x(j)) ^ 2 + (y(i) - y(j)) ^ 2));
        end
    end
end

area = n * (dis / 2) ^ 2 * pi;
disp("面积为:" + num2str(area));

subplot(1, 2, 2);
R = dis / 2; % set R to 0.5
L = max(x) + 2*R;
W = max(y) + 2*R;
for i = 1:n
    if mod(i, 2) == 1
       rectangle('Position', [x(i)-R, y(i)-R, 2*R, 2*R], 'Curvature', [1, 1], '
           FaceColor', 'r', 'EdgeColor', 'none');
    else
```

```
        rectangle('Position', [x(i)-R, y(i)-R, 2*R, 2*R], 'Curvature', [1, 1], '
            FaceColor', 'g', 'EdgeColor', 'none');
    end
end

axis([0 L 0 W]);
xlabel("L") ;
ylabel("W") ;
title("树木优化后种植图");
```

D 问题四代码

```
clc ; clear all ;

e1 = 0.5 ;

Area = 300 ;  % 土地的面积
R = 2.5;  % 树的半径 2.35 2.5 2.65
Iter = 3 ;  % 迭代多次取做多



x_and_y_position = [] ; % 最大数量下的各个点的坐标
n_max = 0 ;  % 二分答案的最大圆的数量

L_t = 0 ;
W_t = 0 ;
for L = 2*R:0.5:Area/2
    W = Area / L ;
    if W < 2*R
        continue ;
    end
    n_left = max(0, floor(L / (2 * R)) * floor(W / (2 * R)) - 100) ; % 圆形的最小数量
    n_right = ceil(L * W / (pi * R * R)) ;  % 圆形的最大数量

    % 因为圆形的数量 满足单调性, 运用二分算法求解满足特定土地的最大圆形数量
    for i = 1:Iter
        left = max(n_max, n_left) ;
```

```
        right = n_right ;
        while left <= right
            mid = floor((left + right) / 2) ;
            if mid <= 0
                continue ;
            end
            disp("现在正在计算: " + num2str(mid)) ;
            [result, solution] = check(mid, L, W, R, e1) ;
            if result == 1
                n_max = mid ; L_t = L ; W_t = W ;
                left = mid + 1 ;
                x_and_y_position = solution ;
                disp(num2str(mid) + "个圆形可以容纳下!") ;
            else
                right = mid - 1 ;
                disp(num2str(mid) + "个圆形不能容纳下!") ;
            end
        end
    end
end

% 绘制圆形
x = x_and_y_position(1:n_max) ;
y = x_and_y_position(n_max+1:end) ;
figure ;
for i = 1:n_max
    rectangle('Position', [x(i)-R, y(i)-R, 2*R, 2*R], 'Curvature', [1, 1], 'FaceColor',
        'r', 'EdgeColor', 'none');
end
axis([0 L_t 0 W_t]);

disp("此时的长: "+L_t+"  宽: "+W_t) ;


% 二分函数的check函数
function [result, solution] = check(m, L, W, R, e1)

    % 初始化迭代点
    x = rand(m, 1) * L ;
```

```
    y = rand(m, 1) * W ;
    X = [x ; y] ;

    % 约束变量的上界和下界
    lb = [repmat(R, m, 1) ; repmat(R, m, 1)] ;
    ub = [repmat(L-R, m, 1) ; repmat(W-R, m, 1)] ;

    % 将固定参数传递给 objfun
    objfun = @(X)objectiveFcn(X,L,W,R) ;

    % 运用模式搜索算法求解函数最小值
    %options = optimoptions('ga','PopulationSize', 50, 'Generations', 100, "HybridFcn
        ","fmincon");
    %[solution,objectiveValue] = ga(objfun,2*m,[],[],[],[],lb,ub,[],options);
    %[solution,objectiveValue] = particleswarm(objfun,2*m,lb,ub);
    %[solution,objectiveValue] = patternsearch(objfun,X,[],[],[],[],lb,ub,[],options);
    options = optimoptions("fmincon","MaxFunctionEvaluations",10000);
    [solution,objectiveValue] = fmincon(objfun,X,[],[],[],[],lb,ub,[],options);

    % 返回check的结果, 表示
    result = objectiveValue < e1 ;
end

function f = objectiveFcn(X, L_block, W_block, R)
    n = size(X, 1) / 2 ;
    x = X(1:n) ;
    y = X(n+1:end) ;
    d = zeros(n, n+4) ;
    for i = 1:n
        for j = 1:n
            if i == j
                continue ;
            end
            tp = sqrt((x(i)-x(j))^2 + (y(i)-y(j))^2) ;
            d(i,j) = smooth(2*R - tp) ;
        end
        d(i, n+1) = smooth(R-y(i)) ;
        d(i, n+2) = smooth(R-x(i)) ;
        d(i, n+3) = smooth(R-W_block+y(i)) ;
```

```
        d(i, n+4) = smooth(R-L_block+x(i)) ;
    end
    %disp(d) ;
    f = sum(sum(d.^2)) ;
end

function f = smooth(a)
    f = a ./ (1 + exp(-a*10000)) ;
end
```