浙江大学第二十二届
大学生数学建模竞赛

2024 年 5 月 13 日 – 5 月 22 日

团队名 7873

题　目　　A　　B

(在所选题目上打勾)

	参赛队员 1	参赛队员 2	参赛队员 3
姓名	戚铭宇	陈景颖	杨云龙
学号	3220102986	3220101920	3220101824
院（系）	数学科学学院	管理学院	经济学院
专业	统计学	信息管理与信息系统	金融学
手机	15988288007	19357571847	18611105404
Email	2918797359@qq.com	18357571847@163.com	3220101824@zju.edu.cn

浙江大学本科生院

浙江大学数学建模实践基

基于模拟退火算法的无人机协同区域搜索问题研究

摘要

无人机协同区域搜索是一种高效、灵活、低成本且安全的搜索方法，能够在复杂或危险的环境中快速覆盖大面积区域，并通过实时数据传输提供即时的情报支持。本文研究了无人机区域协同搜索的路径规划问题，利用模拟退火算法求解出无人机协同搜索的最优路径，并对其效果进行评价。

针对问题一：首先，考虑无人机的搜索规则与搜索地图中的障碍物信息，**将初始地图栅格化**。然后，利用无人机匀速飞行和目标点位置未知等条件，将无人机搜索目标最多的问题转化为搜索面积最大化和搜索路径重复程度最小化的问题。在**建立多目标优化模型**时，**考虑了转向耗时**、任务时间约束、避碰约束和避障约束等条件。最后，利用**模拟退火算法**通过多次迭代，求解出全局最优解下的无人机具体优化路径及其搜索覆盖率和路径重复率。在具体的两个实例中，我们建立了一万平方米的正方形地图，**随机分配障碍物和无人机起点**，分别选择 2 架和 6 架无人机进行求解。受制于障碍物的影响, 场景一、二的搜索覆盖率分别为 12.00%和 22.56%, 路径重复率分别为 0.00%和 3.38%。

针对问题二：首先，建立平面直角坐标系并选定原点为所有无人机的起飞点。然后，利用无人机匀速飞行和总任务时间固定的条件，在不考虑无人机转向耗时的情况下，将约束条件转化为无人机飞行路径的最大限制。接着，根据地图中目标点位置与权重已知的信息，利用路径限制和目标点访问次数限制等条件，建立最大化搜索权重和的**单目标优化模型**。随后，利用**模拟退火算法**求解出无人机的最优路径。为评估模型效果，我们计算了无人机捕获的权重占总权重的比例（优化效率），并通过**多次运行模型进行统计分析**。在具体实例中，场景一选择起点在正方形地图左下角的 2 架无人机搜索 12 个目标点; 场景二选择起点在更大正方形中心的 6 架无人机搜索 200 个目标点。场景一、二的优化效率分别为 0.911422 和 0.680048。

本文的特色在于建立了清晰明了的目标优化模型，通过匀速飞行设定简化约束条件，使复杂问题能够通过简化模型求解。此外，通过模拟退火算法的应用，引入了随机扰动机制，能够有效避免局部最优解，找到全局最优解。此方法兼具简单性和较短的求解运算时间，同时在面对复杂问题时，通过多次迭代实现精确求解，为无人机在不同目的的区域协同搜索中的路径规划提供了宝贵的参考依据。

关键词：无人机区域协同搜索 优化模型 模拟退火算法 栅格法

一、 问题重述

无人机搜索具有灵活性、高效性、低成本和安全性等优点，能够在复杂或危险的环境中快速覆盖大面积区域，并通过实时数据传输提供即时的情报支持。其自主性、可编程性和机动性使其适应各种搜索任务，因此被广泛应用于各个领域。然而，单架无人机在执行复杂任务时受限于其体积和负载能力，难以独立完成。因此，多无人机协同区域搜索成为一种重要的解决方案。

多无人机协同搜索通过分布式覆盖、协同通信和任务分工，显著增加了搜索区域的覆盖率和细致程度，提升了整体搜索效率，并在复杂搜索任务中展现出巨大的潜力。规划无人机的最佳搜索路径并协调无人机间的关系以应对不同搜索要求，成为了无人机协同搜索领域的关键问题。本文将建立数学模型并解决以下问题:

在地面上的某个确定区域内存在M个目标物（M >10），并且有N架无人机（$2 \leqslant N \leqslant 6$）协同完成搜索任务。无人机在同一高度$H$上匀速飞行，且当无人机在地面的投影与目标物的距离不超过d时，视为发现该目标。无人机的搜索范围为椭圆形轨道曲线。以下是两个任务要求:

1.如何在固定时间内搜索到最多的目标点数量？假设地图中存在障碍物且目标点的数量和位置均未知，无人机从地图中不同的随机点出发，并且需要考虑飞行转向的耗时。在限定任务时间T的约束条件下，任务要求我们求出能搜索到的最多的目标点数量。

2.如何在固定时间内搜索到最大权重的目标点？ 假设地图中没有任何障碍物，且目标点的位置和权重均已知，无人机从地图中的同一起点出发，并且忽略飞行转向的耗时。在限定的时间T内，任务要求我们求出所能搜索到的最大权重和。

二、 问题分析

2.1 问题一的分析

问题一是一个典型的全覆盖路径规划问题。在此问题中，我们需要为多架无人机规划路径，使其在有限的任务时间内避开地图中的障碍物，尽可能多地发现目标点。由于目标点的位置未知，问题的本质在于如何在有限的时间内，通过多架无人机协同工作，搜索尽可能多的区域。由于无人机匀速飞行，搜索路径的重叠越小，覆盖的区域就越大。因此，我们将目标函数从发现目标点的数量转变为最大化搜索覆盖率和最小化路径重复率，并根据题目要求设定无人机的时间约束、避障约束和碰壁约束等条件，建立目标优化模型。

我们观察到，该模型具有复杂且离散的搜索空间，由于寻求最优解，容易出现多个局部最优解的问题。为了解决这一问题，我们采用模拟退火算法进行求解。最后，我们将从搜索覆盖率和路径重复率两个维度来评估求解结果，以证明我们确实找到了全局最优解。

2.2 问题二的分析

问题二实际上可以看作是一个带有多重约束的多旅行商问题（点对点的路径规划问题）。在这个问题中，我们需要规划多架无人机的路径，使得在有限的时间内，总权重最大化。我们将时间约束转化为最大飞行路径约束，再根据题目确定每个目标点只能被一个无人机访问一次等约束条件，建立起单目标优化模型。我们将无人机捕获的总权重之和作为目标函数，以无人机是否访问目标点

作为决策变量。与问题一相同，由于路径规划的可能性繁杂，计算难度大，并且传统算法存在陷入局部最优解的情况，我们仍旧采用模拟退火算法来实现对此路径规划问题的求解。此外，为了评估模拟退火算法对此目标优化问题的适配性与我们程序的有效性，我们还采用最优解权重占总权重的比例来反应模型的效率。

三、 符号说明

表 1 符号说明表

符号	说明
N	无人机数
M	目标点数
T	规定任务时间
v	无人机速度
C_{past}	问题一已遍历的栅格
d	搜索半径（问题一中也表示栅格边长的一半）
l	栅格地图中正方形网格列数
r	栅格地图中正方形网格行数
k	无人机已搜索过的总栅格数
搜索覆盖率ω	已搜索面积占总地图面积的比重
路径重复率μ	搜索路径重复栅格数占路径栅格总数的比例
D	问题二单个无人机最大飞行距离
w_i	目标物权重
d_j	无人机 j 的路径总距离
x_{ij}	无人机 j 访问目标点 i 的 0-1 变量
优化效率η	所有无人机捕获的总权重占目标点总权重的比例

四、 模型的建立与求解

4.1 问题一模型的建立与求解

4.1.1 模型准备

使用栅格法将地图初始化。由于题目中规定的搜索区域并非无人机的可飞行区域，因此可以将实际不规则地图填充为使损耗面积最小化的规则矩形，以便规划无人机路径。根据无人机搜索半径 d，我们将地图划分为大小相同的n个单位为$grid_size = d$的正方形。以地图的左下角为原点在第一象限建立平面直角坐标系，用(x_i, y_i)坐标表示每个栅格的中心位置。

$$\begin{cases} x_i = (l-1) \times 2 \times grid\_size + grid\_size \\ y_i = (r-1) \times 2 \times grid\_size + grid\_size \end{cases}$$

如图 1 所示，假定无人机的搜索范围为以自身为中心，半径为 d，高度为 2H 的圆柱形区域。在长宽均为 20 个单位的栅格地图中移动时可搜索范围如图 2 所示。因此可将无人机的搜索区域近似视为铺满整个栅格。

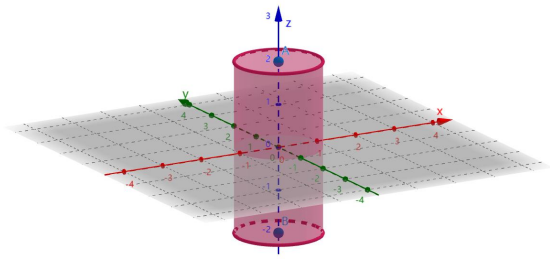

图 1 静态无人机搜索区域

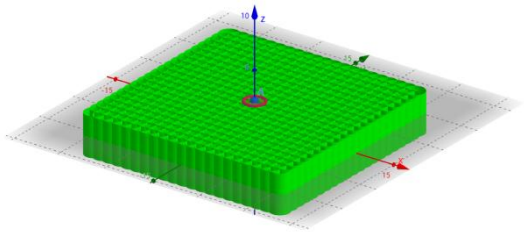

图 2 动态无人机搜索区域

无人机飞行高度恒为 H，将任何高度大于 H 的建筑物或山体视为障碍物。因此可以将栅格分为自由栅格，重复栅格和障碍及遍历栅格三类，用示性函数 $\mathrm{F}(x_i, y_i)$指示栅格的属性。如图 3 所示，白色区域为自由栅格，灰色区域为随机设置的障碍栅格。

$$\mathrm{F}(x_i, y_i) = \begin{cases} 0, & \text{自由栅格} \\ 1, & \text{障碍及遍历栅格} \\ 2, & \text{重复栅格} \end{cases}$$

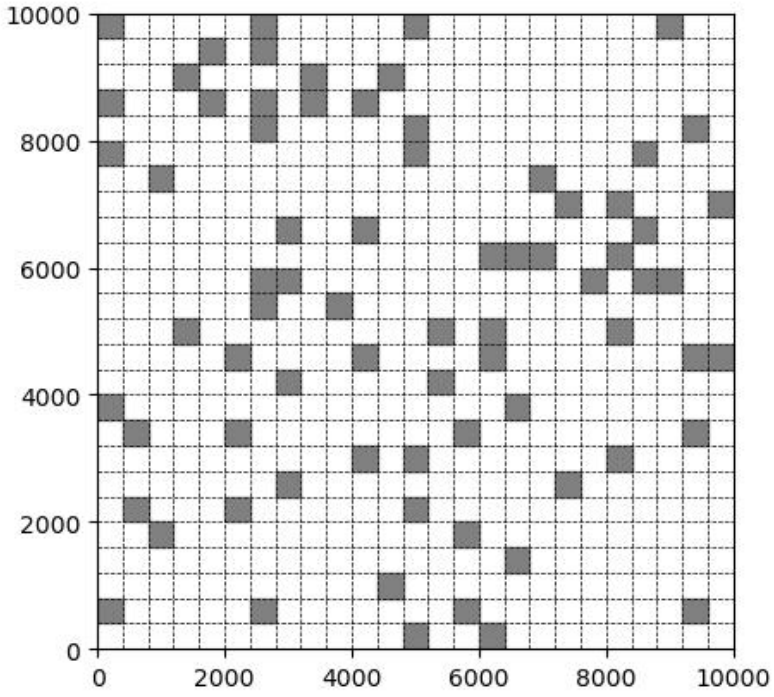

图 3 初始栅格地图示意图

将栅格地图的实时状态记为C_{all}，障碍栅格及已遍历栅格记为C_{past}，重叠栅格记为$C_{overlap}$,，初始化栅格记为C_0。

$$C_{all} = \begin{pmatrix} F(x_1, y_1) & \cdots & F(x_l, y_l) \\ \vdots & \ddots & \vdots \\ F(x_{n-l+1}, y_{n-l+1}) & \cdots & F(x_n, y_n) \end{pmatrix}$$

$$C_{past} = \{(x_i, y_i) \in C_{all} | F(x_i, y_i) = 1\}$$

$$C_{overlap} = \{(x_i, y_i) \in C_0 | F(x_i, y_i) = 2\}$$

4.1.2 4.1.2 模型假设

(1) 场景定义：

目标点不与障碍物发生重合。

搜索区域并非边界不可行的凹多边形，即不存在以下情况：

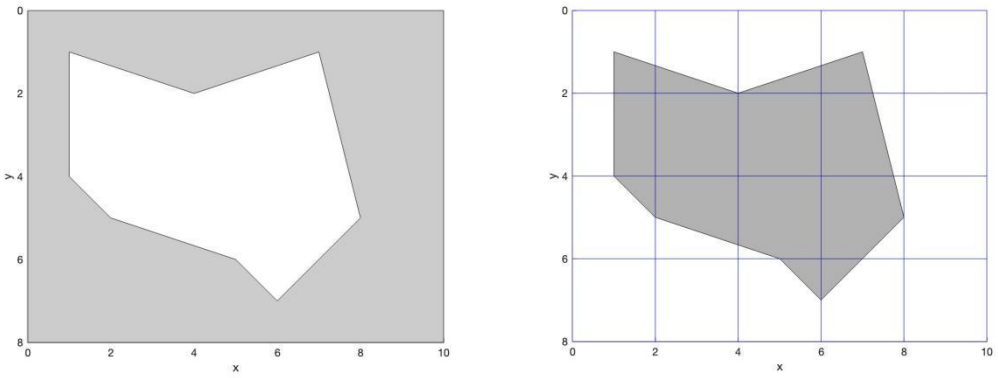

图 4 不被考虑的凹多边形示意图

(2) 无人机属性定义：

无人机每转向 90°所耗费的时间为 t。

无人机只能在栅格中心连线的路径上飞行。

无人机匀速直线飞行，不受风向与风速的影响，速度为 v。

不考虑无人机摄像头质量和装置位置。

所有无人机的性能一致，且均能支撑其行驶完规划路线。

(3) 任务描述：

所有无人机都从无需从同一起点出发，也无需从同一点回收。

视最后一架无人机行驶完其路径所用时间为完成任务所需的最长时间t_{max}。若t_{max}大于规定时间 T，则视为任务失败，记录此时的栅格地图状态。

4.1.3 4.1.3 模型构建

在规定时间内，为了最大化无人机发现目标的数量，即无人机所覆盖栅格地图面积，需要求解目标函数如下：

$$Max \sum_{i=1}^{M} [1 - \prod_{j=1}^{N} (1 - I_{ij})]$$

其中，I_{ij}为 0-1 变量，表示第 j 架无人机是否发现第 i 个目标点（若发现则值为 1）。由于目标点的位置未知，我们合理地假设目标点在地图中均匀分布。因此，求解无人机发现最多目标数的问题可以简化为在规定时间内使所有无人机的搜索面积最大化，即使无人机的搜索覆盖率最大化。

$$搜索覆盖率\omega = \frac{已搜索区域总面积}{地图区域面积-障碍区域面积} \times 100\% = \frac{A_{covered}}{L \times R - A_{barrier}} \times 100\% \quad (1)$$

其中，L 和 R 分别代表地图区域的长和宽。$A_{covered}$为已搜索区域总面积，$A_{barrier}$为障碍区域面积。

$$\begin{aligned} L &= grid\_size \times l \\ R &= grid\_size \times r \\ A_{covered} &= k \times grid\_size^2 \end{aligned} \quad (2)$$

公式中，k代表无人机已搜索过的总栅格数，$grid_size^2$为每个栅格的面积。

在此问题中，由于存在总任务时间的约束，并且我们设定无人机匀速飞行，因此无人机飞行路径重叠越少，说明其能够搜索到更多的区域，从而增加搜索覆盖率。然而，由于受限于总任务时间，不同场景中的搜索覆盖率会有所不同，无法直接进行比较（任务时间长的场景自然搜索覆盖率会高）。因此，为了提供算法效率指标并更清晰地展示解法的性能，定义路径重复率为：

$$\text{路径重复率}\mu = \frac{\text{重复路径栅格数}}{\text{已遍历栅格数}} \tag{3}$$

经过以上分析，我们建立如下模型：

- 目标函数：

$$\text{Max } \omega \quad \& \quad \text{Min } \mu$$

- 约束条件：

1) 避障约束（由代码实现）：无人机只走值为 0 的格，不走值为 1 的格，即$\Delta d_{ik} \geq d$，其中$\Delta d_{ik}=\sqrt{(x_i - x_k)^2 + (y_i - y_k)^2}$，$(x_i, y_i)$为无人机位置，$(x_k, y_k)$为障碍物位置。
2)
3) 避碰约束：己方无人机之间不得相撞，二者需要保持距离，即$\Delta d_{ij} \geq 2d$，其中$\Delta d_{ij}=\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$，$i$、$j$ 分别表示第 i 和第 j 架无人机，$i \neq j$。
4) 时间约束：假设转向 90°的次数为 S，则$\frac{(C_{past}-2)\times 2d}{v} + \frac{2d}{v} + St \leq T$
5) 无人机数量即分解路径约束：设分解路径数为C_{path}，则$C_{past} \in [2,6]$

4.1.4 4.1.4 模型求解

问题一的目标是在规定时间内搜索最多的区域（发现最多的目标）。这个问题本质上是一个组合优化问题，需要在多架无人机的飞行路径上进行全局优化，以最大化目标物的发现数量。然而在本题中，无人机的路径选择是离散的，且组合数极多，再加上多架无人机一起搜索，形成一个庞大的搜索空间。此外，不同路径组合可能会形成多个局部最优解，而不是一个单一的最优解。因此我们使用模拟退火算法来求解，通过再搜索过程中允许一定程度的“坏”解来跳出局部最优解，去找到全局最优解。模拟退火算法逻辑如图 5 所示。

算法步骤——模拟退火算法

Step1：初始化

将地图分为 **l**×**r** 个单位长度为 **grid_size** 的正方形栅格，随机设置若干个障碍物。初始化每架无人机的起始点和路径。设置初始温度$T_{initial}$，最低温度T_{final}，冷却率α与迭代次数G等退火相关参数。

Step2：扰动迭代

通过随机选择方向与随机坐标的双重扰动从当前最优解的领域中选择一个新解。计算并对比当前最优解解和新解的目标函数值。如果：

- 当前解比新解更优（ω更大且μ更小），则接受新解作为当前最优解。
- 如果新解比当前最优解差，则以一定的概率接受新解，概率由 **Metropolis** 准则决定：$P(\text{accept}) = \exp(\frac{\Delta E}{T})$。其中，$\Delta E$为当前最优解解与新解的$\omega$差值，T 为当前温度。

Step3：温度更新

温度以指数降温策略逐步降低：$T = \alpha \times T$。

Step 4：终止

当温度降低到最低温度时，算法终止。输出当前最优解为全局最优解。

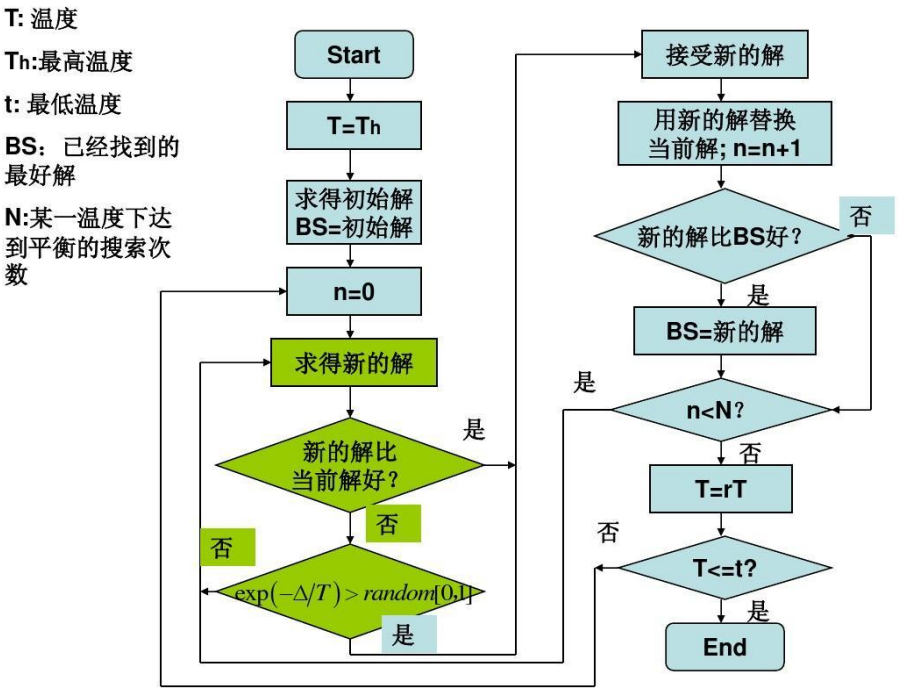

图 5 模拟退火过程示意图

4.1.5 场景一的构建与求解

场景设置

在场景一中我们考虑一个 10000m × 10000m 的正方形地图，并为不同无人机随机设置起点。具体的参数设置如下：

- 无人机数量$N = 2$
- 任务时间$T = 10000$秒
- 无人机速度$v = 20m/s$
- 无人机转弯耗时$t = 3s$
- 搜索半径$d = 200$米

模拟退火参数设置

在模拟退火算法中，模型在扰动下不断进行迭代，温度逐步降低，直到达到最低温度。具体的算法如下：

- 初始温度$T_{initial} = 1000$
- 冷却率$\alpha = 0.99$
- 最低温度$T_{final} = 0.001$

结果展示

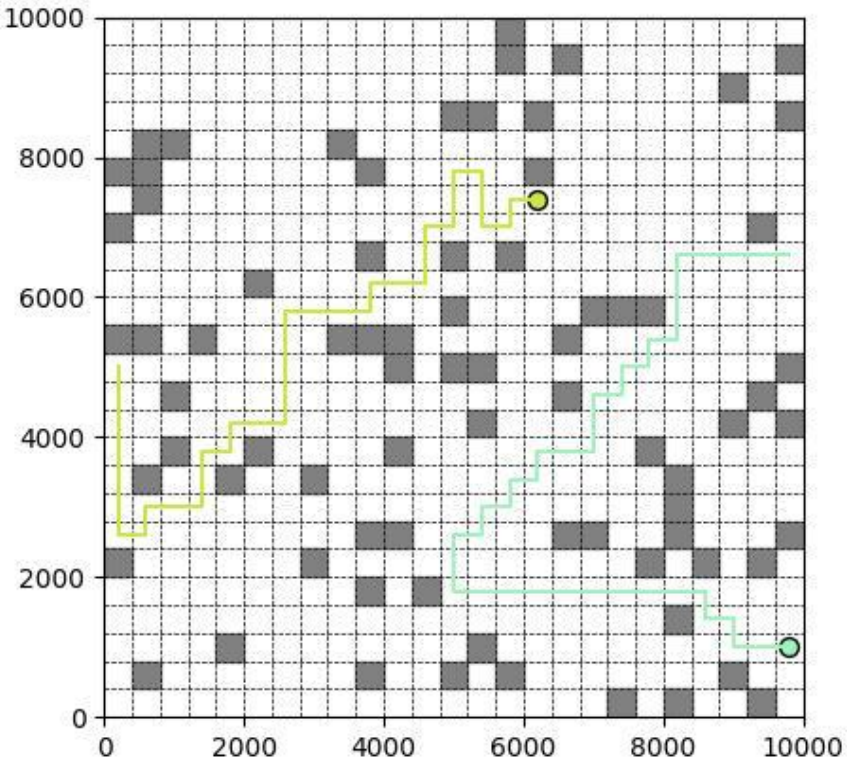

图 1 场景一无人机最优搜索路径示意图

表 2 场景一无人机最优搜索路径坐标

无人机编号	无人机搜索路径
1	(9800,1000) -> (9000,1000) -> (9000,1400) -> (8600,1400) -> (8600,1800) -> (5000,1800) -> (5000,2600) -> (5400,2600) -> (5400,3000) -> (5800,3000) -> (5800,3400) -> (6200,3400) -> (6200,3800) -> (7000,3800) -> (7000,4600) -> (7400,4600) -> (7400,5000) -> (7800,5000) -> (7800,5400) -> (8200,5400) -> (8200,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600) -> (9800,6600)
2	(6200,7400) -> (5800,7400) -> (5800,7000) -> (5400,7000) -> (5400,7800) -> (5000,7800) -> (5000,7000) -> (4600,7000) -> (4600,6200) -> (3800,6200) -> (3800,5800) -> (2600,5800) -> (2600,4200) -> (1800,4200) -> (1800,3800) -> (1400,3800) -> (1400,3000) -> (600,3000) -> (600,2600) -> (200,2600) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000) -> (200,5000)

图 4 中，黄色和浅绿色线分别表示两架无人机的搜索路径，其中黄色线为 1 号无人机的飞行路径，浅绿色线为 2 号无人机的飞行路径。圆点为无人机起点。在任务执行过程中，两架无人机均飞行 780 秒。搜索覆盖率为 12.00%，路径重复率为 0.00%。由于任务时间有限且场地中有障碍物阻拦，搜索覆盖率本就不会太大。然而，路径重复率为 0，这表明当前解已经相当理想。

4.1.6 场景二的构建与求解

场景设置

我们考虑一个更为复杂的 10000m × 10000m 的正方形地图与情景，其中具体参数设置如下：

- 无人机数量$N = 6$
- 任务时间$T = 10000$秒
- 无人机速度$v = 20m/s$
- 无人机转弯耗时$t = 3s$
- 搜索半径$d = 200$米

模拟退火参数设置

在场景二中我们依旧采取与场景一相同的模拟退火参数设置，具体如下：

- 初始温度$T_{initial} = 1000$
- 冷却率$\alpha = 0.99$
- 最低温度$T_{final} = 0.001$

结果展示

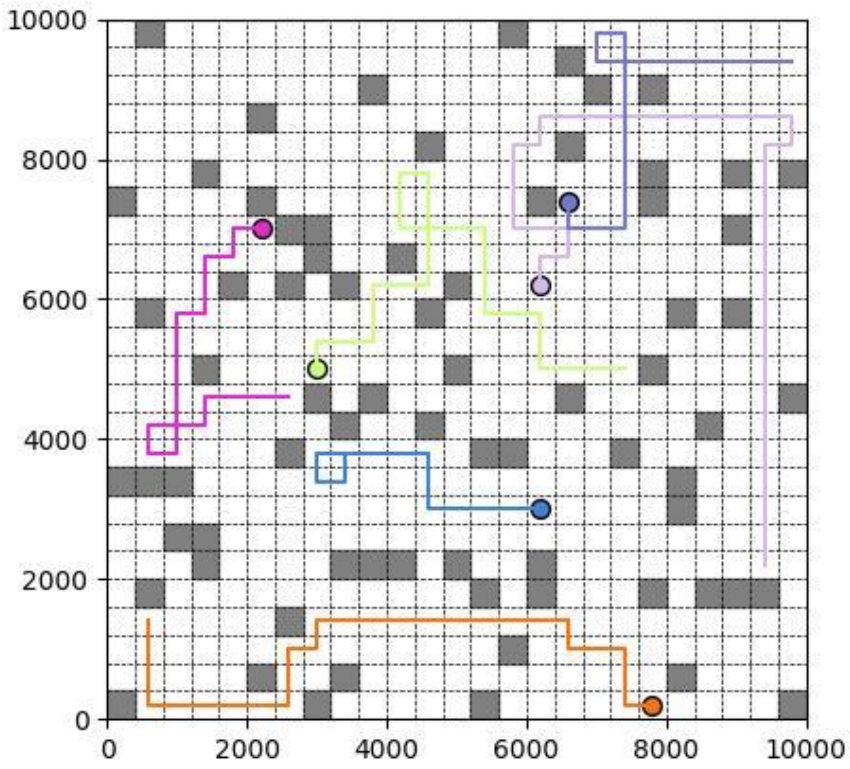

图 2 场景二无人机路径示意图

图中，六种颜色分别代表六架无人机的飞行路线，对应颜色的圆点表示该架无人机的起点。有关于具体无人机飞行路径的坐标请见附录部分。

表 3 场景二无人机搜索结果

项目	1 号无人机	2 号无人机	3 号无人机	4 号无人机	5 号无人机	6 号无人机
飞行时间/s	393	393	393	393	393	393
ω/%	22.56					
μ/%	3.38					

受到场地上存在障碍物以及时间约束的限制，仅有六个随机分布的起飞点的无人机难以完全搜索所有区域，因此结果可能呈现较低的状态。然而，观察表明，路径重复率较低，仍在我们可以接受的范围内。因此，我们有理由相信该解已经相对较优。

4.2 问题二模型的建立与求解

4.2.1 模型准备

我们将根据地图类型选取原点建立平面直角坐标系，将目标物在坐标系中的位置以坐标(x_k, y_k)表示。

我们假设:

(1) 地图中不存在任何障碍物。

(2) 无人机匀速飞行，速度为 v。T 时间内无人机的最大飞行距离为$D = v \times T$。

(3) 无人机从同一点（坐标系原点）起飞，但无需从同一点收回。

(4) 目标点位置不发生变化。

(5) 不考虑边界对无人机的阻碍作用，即无人机在边界外仍可飞行。

4.2.2 模型建立

决策变量：令x_{ij}为 0-1 变量，如果无人机 j 访问了目标点 i，则$x_{ij} = 1$,否则为 0。

目标函数：最大化无人机访问目标点的总权重

$$\mathrm{Max}\sum_{i=1}^{M}\sum_{j=1}^{N} w_i \cdot x_{ij}$$

约束条件:

1) 每个目标点只能被一架无人机访问一次

$$\sum_{j}^{N} x_{ij} \le 1,\ \forall i = 1,2,\ldots,M \tag{4}$$

2) 每架无人机飞行的总距离不超过最大飞行距离

$$d_j \le D,\ \forall j = 1,2,\ldots,N \tag{5}$$

3) 决策变量的二进制性

$$x_{ij} \in \{0,1\} \quad \forall i = 1,2,\ldots,M,\ \forall j = 1,2,\ldots,N$$

4.2.3 模型的求解——模拟退火算法

观察到这是一多目标优化的旅行商问题，其主要的目标是使多架无人机在规定时间内访问目标点权重最大。经我们转化后将问题变为了在每架无人机飞行长度小于给定最大距离情况下访问总目标点的权重最大化问题。在无人机路径规划问题中可能出现多个局部最优解，为了防止算法陷入局部最优解并减少计算的复杂程度，我们使用模拟退火算法来求解，通过随机扰动来跳出局部最优解，向全局最优解逼近。

1. 初始化：使每架无人机访问其自身编号的目标点来作为一个初始解。
2. 迭代：在每一代中，我们通过不停扰动形成新解，并根据以下条件判断是否接受新解。
   - 如果新解的总权重更大，接受新解。
   - 如果新解的总权重相等但路径更短，接受新解。
   - 如果新解的总权重相等但路径更长，以一定概率接受新解（与当前温度有关）。概率 $P = \exp(\frac{\sum \text{best}_{\text{distance}} - \sum \text{new}_{\text{distance}}}{\text{tempreture}})$
   - 如果新解的总权重更小，以一定概率接受新解（与当前温度有关）。概率为 $P = \exp(\frac{\text{new}_{\text{weight}} - \text{best}_{\text{weight}}}{\text{tempreture}})$
3. 温度更新：逐渐降低温度，直至达到预设的最小温度或最大迭代次数，以使得算法收敛到全局最优解的可能性增加。
4. 最优解：取当前解为最优解并输出，随后可视化最优路径规划结果。

对于扰动，我们用三种方法来构建扰动函数：

1) 交换目标点：随机选择两架无人机并交换其访问的一个目标点。
2) 添加未访问的目标点：随机选择一架无人机并为其添加一个未访问的目标点。
3) 替换为未访问的目标点：随机选择一架无人机并用一个未访问的目标点替换其现有的目标点。

4.2.4 模型评估

为了更直观地反映我们模型的效果与路径规划的优化程度，我们采用优化效率（最优解权重占总权重的比例）来评估算法的表现。优化效率的定义如下：

$$\text{优化效率}\eta = \frac{\text{所有无人机收集到的权重之和}}{\text{所有目标点的总权重}} \tag{6}$$

在每次求解最优路径后中，我们对优化效率和迭代次数作图，将优化效率作为 y 轴，迭代次数作为 x 轴，以此来反应优化效率与迭代次数的关系，从而找到优化效率的收敛值。通过该值，我们可以评估模拟退火算法对此目标优化问题的适配性与我们程序的有效性。

4.2.5 场景一的构建与求解

场景设置

在场景一中，我们考虑一个 $100\text{m} \times 100\text{m}$ 的正方形地图，并选择正方形的一角作为原点建立平面直角坐标系（无人机从原点出发）。假设有 2 架无人机和 12 个目标点。通过程序随机生成目标点的位置和权重，并将这些信息视为已知数据。具体参数如下：

- 目标点$M = 12$
- 无人机数量$N = 2$

- 任务时间$T = 130$秒
- 无人机速度$v = 1\ m/s$
- 搜索半径$d = 5$米

因此，每架无人机飞行的最大飞行距离$D = v \times T = 130$米

模拟退火算法的设置

为了优化无人机的路径规划，我们采用模拟退火算法。该算法的具体参数设置如下

- 迭代次数$G = 1000$
- 初始温度$T_{initial} = 100$
- 冷却率$\alpha = 0.99$
- 最低温度$T_{final} = 0.1$

结果展示

目标物的散点图与最优无人机路线图如下所示。图中，红色圆点表示目标物，蓝色圆点表示起点，目标物半径越大代表其权重越大，目标物旁的数字代表目标物编号。最优路径用绿色线条表示。

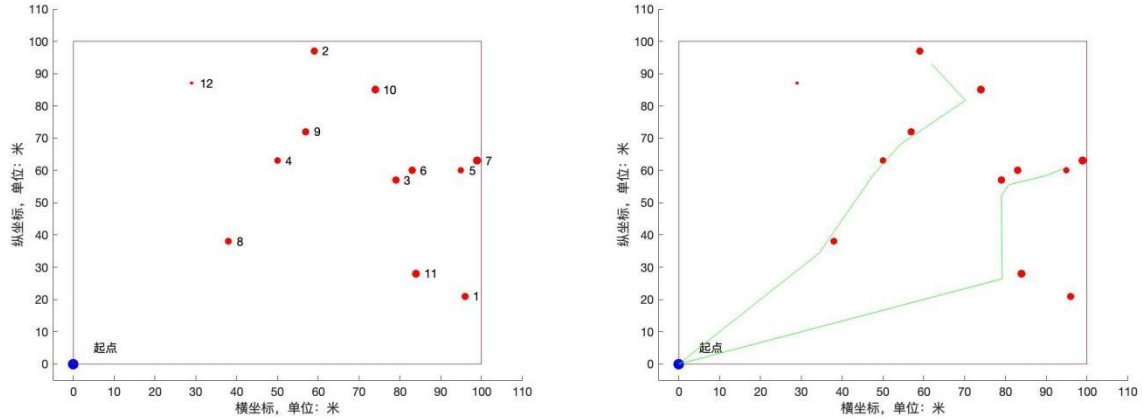

图 3 目标物散点图与最优无人机路线图

表 4 场景一目标物坐标与权重表

目标点编号	x 坐标	y 坐标	权重
1	96	21	74
2	59	97	76
3	79	57	81
4	50	63	59
5	95	60	54
6	83	60	82
7	99	63	100
8	38	38	68
9	57	72	77
10	74	85	91
11	84	28	94
12	29	87	2

表 5 场景一最优路径

无人机编号	M_{catch}	X_{catch}	Y_{catch}

1 号无人机	0	0	0
	11	79.25658351	26.41886117
	3	79.04194979	52.00017598
	6	80.78271238	55.51852305
	5	90.23129823	58.49684218
	7	94.55222901	60.71585176
2 号无人机	0	0	0
	8	34.46446609	34.46446609
	4	47.60921553	58.60862782
	9	54.12923756	67.90625806
	10	70.20953485	81.73926789
	2	61.95996516	92.97028459

表 2 和表 3 分别列出了地图中所有目标点的位置和权重，以及在该场景中最优路径捕获的目标点。在表 3 中，M_{catch}表示最优路径中捕获的目标点编号，X_{catch}表示最优路径经过并能捕获目标点的位置的横坐标，Y_{catch}表示其纵坐标。

模型评估

我们发现，在经过 32 次迭代后，优化效率已经收敛到稳定值 0.911422，并且不再发生变化。这表明路径已经达到了模拟退火算法所能计算出的最优解。从优化效率的收敛值 0.911422 来看，无人机已经捕获了最大可能捕获的权重，占所有权重值的 91%。由此可见，我们找到的路径确实是无人机的最优路径。

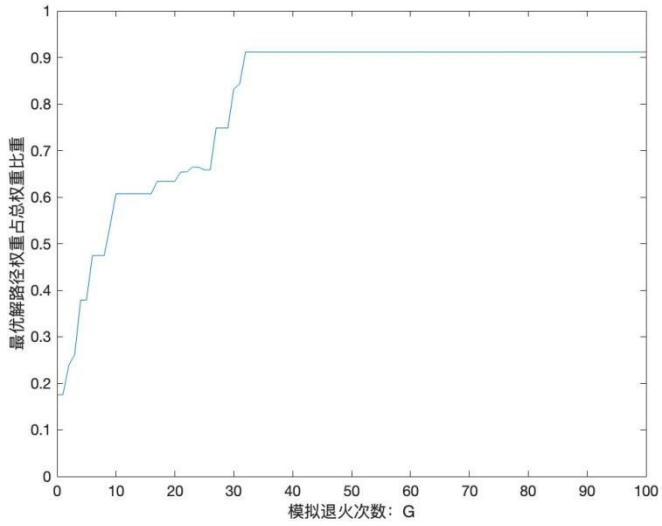

图 4 场景一模拟退火算法的效率

4.2.6 场景二的构建与求解

场景设置

在此实例中，我们考虑一个包含 200 个目标点 $1000\mathrm{m} \times 1000\mathrm{m}$ 的正方形，以正方形中心点为原点，建立平面直角坐标系（无人机从原点起飞）。我们的目标是利用 6 架无人机在给定的最大飞行距离内，规划各自的飞行路径，以最大化总目标权重的覆盖率。具体设定与参数如下：

- 目标点$M = 200$

- 无人机数量$N = 6$
- 任务时间$T = 1100$秒
- 无人机速度$v = 1\ m/s$
- 无人机最大搜索距离$D = v \times T = 1100$米
- 搜索半径$d = 5$米

模拟退火算法的设置

为了能更好地应用于复杂化的场景并找到最优的飞机路径，我们将迭代次数增加了一倍。具体参数如下：

- 迭代次数$G = 200000$
- 初始温度$T_{initial} = 100$
- 冷却率$\alpha = 0.99995$
- 最低温度$T_{final} = 0.001$

结果展示

目标物的散点图与最优无人机路线图 6、7 所示。图中，红色圆点表示目标物，蓝色圆点表示起点，目标物半径越大代表其权重越大，目标物旁的数字代表目标物编号。最优路径用绿色线条表示。

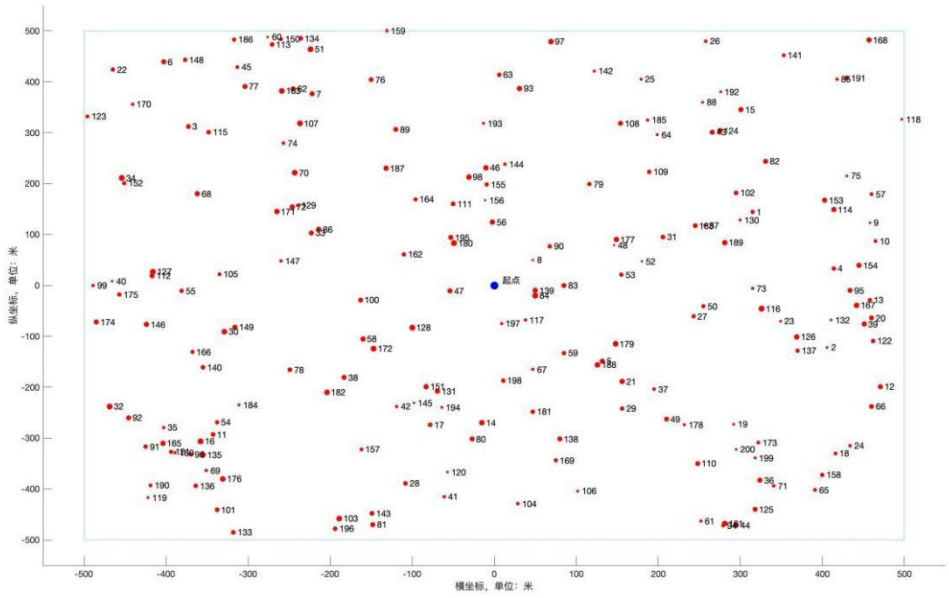

图 5 场景二目标点的散点图

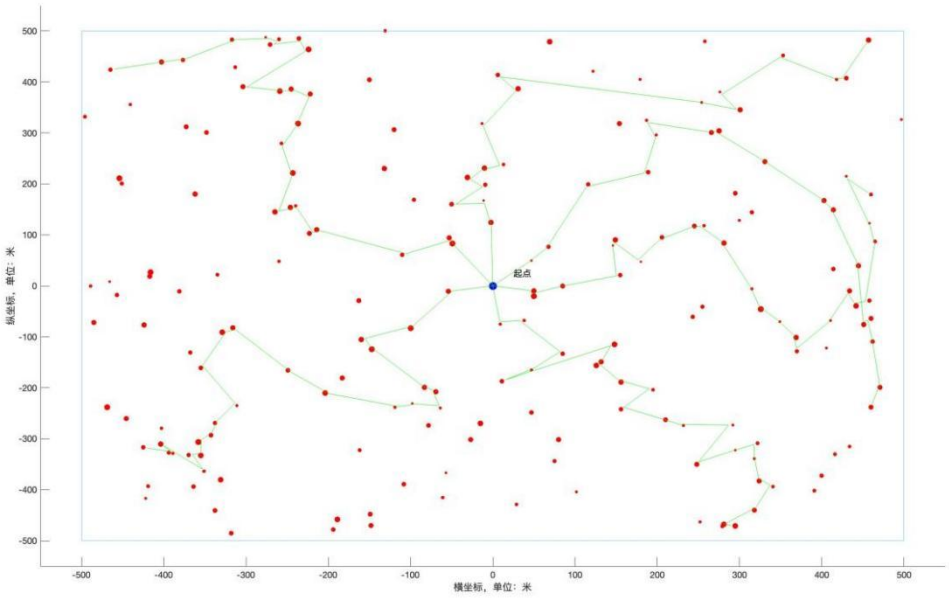

图 6 场景二最优路径示意图

有关场景二具体的目标点的位置和权重，以及最优路径捕获的目标点的内容请参见附录。

模型评估

我们对优化效率和迭代次数作图，结果如下图所示：

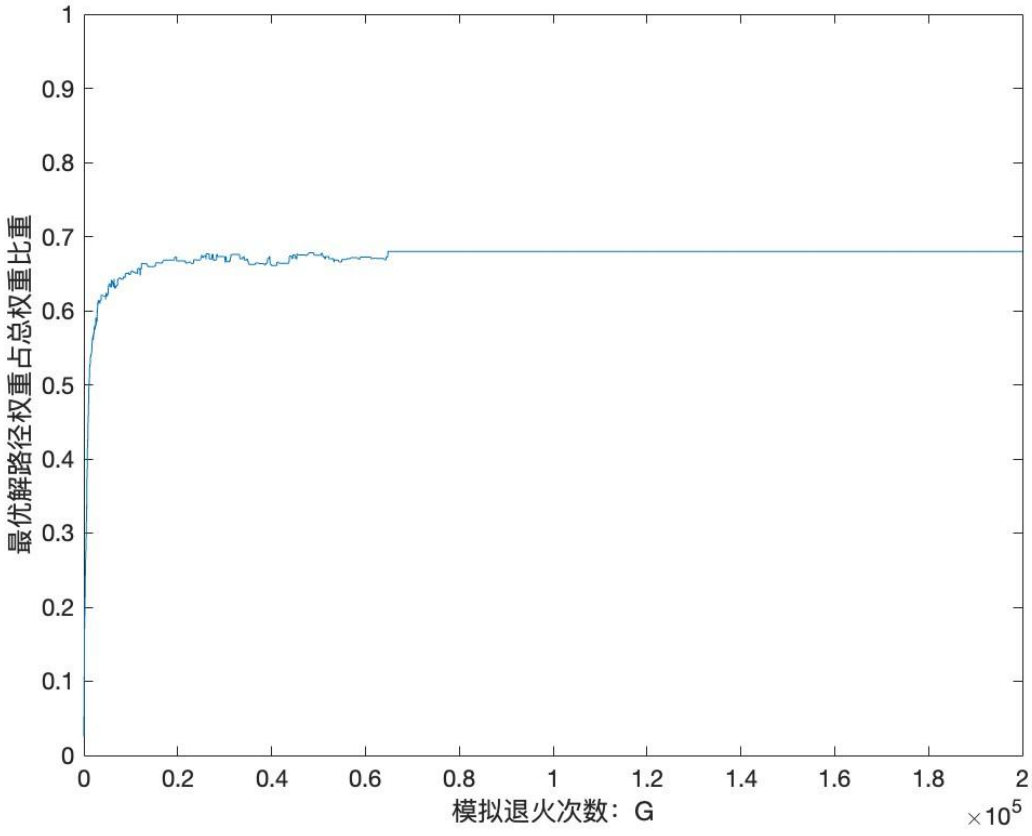

图 7 场景二模拟退火算法的效率

从图中可以看出，在迭代次数达到 80,000 次之后，优化效率基本不再增加，稳定在 0.680048。说明路径已经达到了模拟退火算法所能计算出的最优解，也体现了我们模型与解法的优越性。此外，我们运行了 20 次算法，统计出了模型的最大优化效率、最小优化效率、均值和方差。具体结果如下：

表 6 算法多次运行结果

类别	优化效率				
原始数据	0.664353	0.666767	0.629138	0.680048	0.657108
	0.659221	0.669786	0.641111	0.664353	0.674716
	0.669283	0.5901	0.647852	0.621793	0.62954
	0.616963	0.62793	0.610021	0.665962	0.677634
均值			0.64818395		
方差			0.000660853		

结果显示，模拟退火算法在多次运行中表现出较为稳定的优化效果，并在一定的迭代次数后达到最优解（200,000 次之后都收敛为稳定值）。这些结果证明了模拟退火算法在多无人机路径规划问题中的有效性。

五、 模型的评价与推广

5.1 模型的优点

(1) 在分析无人机搜索目标数量最多的问题时，本文采用了栅格法。其通过将空间划分为规则的网格，使得空间搜索和导航问题可以转化为网格上的操作，简化了复杂环境的表示和处理，使算法的实现更加直观和简洁，提高了数据处理效率。

(2) 在分析无人机捕获最大权重的路径规划问题时，本文将时间约束转化为最大路程约束，建立了单目标优化模型进行求解。

(3) 为了应对问题的复杂性，本文采用了模拟退火算法，使问题简化，并找到全局最优解，避免了陷入局部解。

(4) 文章通过建立优化效率指标，对算法效果进行了评估，并通过可视化方法展示了所得最优解的稳定性。

(5) 在问题一中，本文综合考虑了障碍物约束、无人机不同起点以及飞行转向耗时等因素，使模型更加贴近现实生活，从而便于模型的进一步推广与应用。

(6) 在问题二中，为了评估模型的稳定性，文章对模型进行了多次重复求解。通过数理统计的角度，利用优化效率的均值与方差来说明使用模拟退火算法的合理性和求解模型的优越性。

(7) 在模型构建与求解过程中，使用了 Python、Excel、Matlab 等软件，使得求解过程专业、精确且可靠。

(8) 文章使用多种可视化图表来阐述模型与结果，使各种数据与求解过程更加清晰明了。

5.2 模型的缺陷

(1) 模型可能难以应对复杂情景，对大面积与针对性推广造成困难。
(2) 考虑障碍物使得模型复杂，求解出的搜索覆盖率较高
(3) 当迭代次数增加模拟退火算法运行时间长，增加问题求解耗时。

六、 参考文献

[1] 王兴旺,张清杨,姜守勇,等.基于改进鲸鱼优化算法的动态无人机路径规划[J/OL].计算机应用,1-11[2024-05-21].http://kns.cnki.net/kcms/detail/51.1307.tp.20240509.1511.004.html.
[2] 王兴旺, et al.基于改进鲸鱼优化算法的动态无人机路径规划.计算机应用 1-11.
[3] 王兴旺,张清杨,姜守勇 & 董永权.基于改进鲸鱼优化算法的动态无人机路径规划.计算机应用 1-11.
[4] 江南,徐海芹,邢浩翔.基于 TSACO 及动态避障策略的无人机路径规划[J/OL].计算机应用研究,1-7[2024-05-21].https://doi.org/10.19734/j.issn.1001-3695.2024.01.0028.
[5] 赵畅,刘允刚,陈琳,等.面向元启发式算法的多无人机路径规划现状与展望[J].控制与决策,2022,37(05):1102-1115.DOI:10.13195/j.kzyjc.2021.1210.
[6] 陈星；陈卓；杨博文；李翱翔，基于航迹消除与策略迭代的无人机集群区域目标搜索方法，指挥控制与仿真，2024：46（01），2024
[7] 王伟伦；尤明；孙磊；张秀云；宗群，未知环境下无人机集群智能协同探索路径规划，工程科学学报，2024
[8] 王兴旺；张清杨；姜守勇；董永权，基于改进鲸鱼优化算法的动态无人机路径规划，计算机应用，2024
[9] 唐颂；吴建源，基于改进遗传算法的协同航迹规划方法，电光与控制，2024
[10] 李忠伟；刘旭阳；罗偲；王晓政，旅行商和覆盖路径规划问题的自适应遗传算法，计算机仿真.，2024：41（02），2024
[11] 盛景泰；杜亚男，约束条件下的多无人机协同任务分配方法，应用科技.，2023：50（05）

七、 附录

1.

表 7 问题一场景二无人机搜索路径

无人机编号	无人机搜索路径
1	(3000,5000) -> (3000,5400) -> (3800,5400) -> (3800,6200) -> (4600,6200) -> (4600,7800) -> (4200,7800) -> (4200,7000) -> (5400,7000) -> (5400,5800) -> (6200,5800) -> (6200,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000) -> (7400,5000)
2	(6200,3000) -> (4600,3000) -> (4600,3800) -> (3000,3800) -> (3000,3400) -> (3400,3400) -> (3400,3800) -> (3400,3800) -> (3400,3800) -> (3400,3800) -> (3400,3800) -> (3400,3800) -> (3400,3800) -> (3400,3800)
3	(6200,6200) -> (6200,6600) -> (6600,6600) -> (6600,7000) -> (5800,7000) -> (5800,8200) -> (6200,8200) -> (6200,8600) -> (9800,8600) -> (9800,8200) -> (9400,8200) -> (9400,2200)
4	(6600,7400) -> (6600,7000) -> (7400,7000) -> (7400,9800) -> (7000,9800) -> (7000,9400) -> (9800,9400) -> (9800,9400) -> (9800,9400) -> (9800,9400) -> (9800,9400) -> (9800,9400) -> (9800,9400) -> (9800,9400) -> (9800,9400) -> (9800,9400) -> (9800,9400) -> (9800,9400) -> (9800,9400) -> (9800,9400)
5	(7800,200) -> (7400,200) -> (7400,1000) -> (6600,1000) -> (6600,1400) -> (3000,1400) -> (3000,1000) -> (2600,1000) -> (2600,200) -> (600,200) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400) -> (600,1400)
6	(2200,7000) -> (1800,7000) -> (1800,6600) -> (1400,6600) -> (1400,5800) -> (1000,5800) -> (1000,3800) -> (600,3800) -> (600,4200) -> (1400,4200) -> (1400,4600) -> (2600,4600) -> (2600,4600) -> (2600,4600) -> (2600,4600) -> (2600,4600) -> (2600,4600) -> (2600,4600) -> (2600,4600) -> (2600,4600)

2.

表 8 问题二场景二目标点坐标及权重

目标点编号	x	y	weight	目标点编号	x	y	weight
1	315	144	42	38	-183	-181	67
2	406	-122	9	39	451	-76	70

3	-373	312	60	40	-466	8	6
4	414	33	47	41	-61	-415	25
5	132	-149	70	42	-119	-238	22
6	-403	439	70	43	266	301	67
7	-222	376	64	44	295	-471	85
8	47	50	3	45	-313	429	34
9	458	123	6	46	-10	231	78
10	465	87	32	47	-54	-11	68
11	-343	-293	53	48	146	79	0
12	471	-199	66	49	210	-263	60
13	458	-29	41	50	255	-41	39
14	-15	-270	82	51	-224	464	92
15	301	345	72	52	180	47	0
16	-358	-306	97	53	155	21	46
17	-78	-274	53	54	-338	-269	42
18	416	-330	32	55	-381	-11	46
19	292	-273	10	56	-2	124	77
20	460	-64	61	57	460	179	32
21	156	-189	78	58	-160	-105	79
22	-465	424	42	59	85	-133	47
23	349	-70	9	60	-276	488	3
24	434	-315	26	61	252	-463	17
25	179	405	15	62	-245	386	72
26	258	480	28	63	6	414	47
27	243	-61	44	64	199	296	15
28	-108	-389	53	65	391	-402	34
29	156	-242	46	66	460	-238	61
30	-329	-91	88	67	47	-165	19
31	206	95	52	68	-362	180	74
32	-469	-238	95	69	-351	-364	24
33	-223	103	64	70	-243	221	92
34	-454	211	96	71	341	-394	27
35	-403	-279	24	72	-246	154	77
36	324	-383	68	73	315	-6	19
37	195	-204	29	74	-257	279	29
75	430	215	9	114	414	149	72
76	-150	404	58	115	-348	301	52
77	-304	391	69	116	326	-46	100
78	-249	-166	55	117	38	-68	22
79	116	199	42	118	497	326	10

80	-27	-302	65	119	-422	-417	11
81	-148	-470	65	120	-57	-367	6
82	331	244	68	121	-394	-327	40
83	85	0	64	122	462	-109	45
84	50	-20	95	123	-496	332	36
85	418	405	21	124	275	304	77
86	-214	110	71	125	318	-440	63
87	257	118	23	126	369	-101	77
88	254	360	12	127	-416	27	94
89	-120	306	61	128	-100	-83	98
90	68	77	45	129	-240	157	19
91	-425	-317	46	130	300	128	14
92	-446	-260	66	131	-69	-208	70
93	31	387	77	132	411	-68	9
94	279	-472	35	133	-318	-485	53
95	434	-10	66	134	-236	485	53
96	-370	-332	42	135	-355	-333	86
97	69	479	85	136	-364	-394	48
98	-31	213	84	137	370	-128	39
99	-489	0	25	138	80	-302	67
100	-163	-29	61	139	50	-10	74
101	-338	-441	58	140	-355	-161	52
102	295	182	54	141	353	452	35
103	-189	-458	87	142	122	421	15
104	29	-429	26	143	-149	-448	59
105	-335	22	32	144	13	238	26
106	102	-404	12	145	-98	-231	4
107	-237	318	94	146	-424	-77	76
108	154	318	65	147	-260	48	24
109	189	223	48	148	-377	443	44
110	248	-350	64	149	-316	-82	69
111	-50	160	55	150	-260	484	36
112	-417	19	65	151	-83	-199	74
113	-271	473	54	152	-451	201	39
153	403	167	69	177	149	90	77
154	445	39	71	178	232	-274	16
155	-9	198	44	179	148	-115	87
156	-11	167	1	180	-49	83	99
157	-162	-322	33	181	47	-248	51
158	400	-372	42	182	-204	-210	89

159	-131	500	27	183	245	117	59
160	-389	-329	19	184	-311	-235	15
161	281	-468	82	185	187	325	20
162	-110	61	43	186	-317	483	41
163	-259	382	89	187	-132	230	75
164	-96	169	39	188	126	-156	83
165	-404	-310	77	189	281	84	79
166	-368	-131	40	190	-419	-393	32
167	442	-39	81	191	430	407	53
168	457	482	76	192	276	380	9
169	75	-344	38	193	-13	318	11
170	-441	356	21	194	-64	-240	13
171	-265	145	79	195	-53	94	68
172	-147	-124	95	196	-194	-478	50
173	322	-309	33	197	9	-75	19
174	-485	-72	67	198	11	-187	49
175	-457	-18	44	199	318	-339	14
176	-331	-380	84	200	295	-322	5

3.

表 9 问题二场景二无人机路径

无人机编号	M_{catch}	X_{catch}	Y_{catch}	无人机编号	M_{catch}	X_{catch}	Y_{catch}
1 号	0	0	0	4 号	0	0	0
	8	43.57544996	46.35686166		47	-49.10061726	-10.00197759
	90	64.88352893	73.09006291		128	-97.14020582	-78.89858838
	79	114.1192025	194.3672254		58	-155.3822672	-103.0825686
	109	184.32978	221.2142103		172	-148.8598785	-119.3587877
	64	198.0375272	291.0935098		151	-86.18640002	-195.1468383
	185	188.5477026	320.2455687		131	-73.00409355	-205.0054658
	43	261.1475603	302.2057481		194	-65.24591891	-235.1577189
	124	270.041422	303.357735		145	-93.03980241	-231.6296347
	82	327.4177	247.4881		42	-114.144	-236.808

		419	839			0657	4037
	153	399.5772928	170.6448697		182	-199.2086965	-211.4294793
	114	411.2274581	153.1608907		78	-245.3063854	-169.3700461
	154	443.5816012	43.79459537		149	-312.8549367	-85.88697524
	39	450.690962	-71.00955958		30	-324.2333247	-89.49042841
	20	456.005729	-67.00762345		140	-353.0239095	-156.4070634
	122	461.2934292	-104.050176		184	-313.357643	-230.5907461
	12	470.491508	-194.0259236		54	-335.3000357	-264.791652
	66	461.1603518	-233.136505		11	-341.6833342	-288.1764752
2 号	0	0	0		16	-354.6238023	-302.3120074
	56	-1.919365326	119.0006502		135	-354.9387107	-328.0003757
	156	-10.070573362	162.0871427		96	-365.1674923	-330.7166958
	111	-45.006816662	160.2609976		69	-352.9582889	-359.3994452
	155	-12.451537718	194.3824192		165	-400.4071385	-313.4772613
	98	-27.471047726	209.4578972		160	-391.9608335	-324.9709225
	46	-13.149496675	227.1166161		121	-391.9608335	-324.9709225
	144	8.383848646	236.0787643		91	-420.1394531	-318.1726398
	193	-11.737166666	313.1621025	5 号	0	0	0
	93	28.49530544	382.6725868		139	45.09709662	-9.019419324
	63	8.916361492	409.9386165		84	47.96144819	-15.43444346
	88	249.1006727	360.9982944		83	80.38469119	-1.923258857
	15	296.2218631	346.4728908		53	150.2204699	19.53163636

	192	278.5823874	375.7184961		48	146.3539597	74.01254449
	141	349.5084617	448.4210113		177	148.1835701	85.06710609
	85	413.7771045	407.677154		52	176.7934999	50.83645113
	191	425.00435	407.2085216		31	203.2419382	90.829497
	168	455.0334074	477.4029886		183	240.7632818	114.3447751
3 号	0	0	0		87	252.1220776	116.9018776
	197	8.404273897	-70.03561581		189	277.7017379	87.75785403
	117	33.01178518	-68.34309308		73	313.1518018	-1.354124051
	59	81.86688113	-129.1033904		116	324.6172194	-41.19501114
	67	50.48371709	-161.4133978		23	345.7695686	-66.18367807
	198	15.19599269	-184.2808742		126	366.2248824	-96.84082675
	179	143.5669639	-117.3126156		137	369.3986182	-123.0362978
	5	133.7145107	-144.3031443		132	407.9849857	-71.98869509
	188	128.7528652	-151.826065		95	432.0651171	-14.61044773
	21	153.0441518	-184.9672638		167	440.1137763	-34.36943197
	37	190.4466111	-201.9344131		13	453.2111313	-30.43761506
	29	159.2596636	-238.2086159		10	464.5005895	82.0250036
	49	205.5075542	-260.8050215		9	458.7834406	118.0617593
	178	227.5244065	-271.7708604		75	431.4232141	210.2068318
	19	287.0009083	-272.9046991		57	456.62327	182.6875052
	110	250.2570278	-345.5384055	6 号	0	0	0
	200	290.5749799	-324.3279168		180	-46.45810137	78.69433497

	173	317.5060811	-311.1919609
	199	317.9112053	-334.0007885
	36	323.3834265	-378.0381619
	71	337.2947323	-390.6427703
	125	319.8204448	-435.3431791
	161	284.8262105	-464.781287
	94	282.1402865	-468.1091645
	44	290.1217401	-469.9033778

195	-51.03488921	89.402355
162	-105.4953425	63.16980654
86	-209.4093174	108.018679
33	-218.3095828	104.7320468
129	-238.0835423	152.3818629
72	-241.1012871	152.998695
171	-260.2585189	146.5869332
70	-244.1296583	216.1292842
74	-255.9972394	274.1015848
107	-238.985801	313.4112535
7	-223.3095684	371.1745435
62	-240.8721014	383.1785725
163	-254.0105338	382.3243866
77	-299.0736387	390.1450358
51	-227.5643456	460.4935146
134	-234.3726086	480.2722524
113	-266.0957309	473.9737271
150	-262.5974902	479.7276418
60	-271.7452065	485.373837
186	-312.0068646	483.2619146
148	-372.749496	445.6331001
6	-398.1160331	440.070919

					22	-460.1383732	425.1681546

4. 问题一的 Python 代码（给出了场景二的函数文件，场景一的需要自己变动参数）

```
import numpy as np
import random
import math
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle


# 初始化参数
d = 200  # 目标物的发现距离
N = 6  # 无人机数量
T = 10000  # 任务时间
v = 20  # 无人机的速度
t = 3  # 无人机转弯时间

# 假设搜索区域尺寸
search_area_x = 10000
search_area_y = 10000

# 全局变量来跟踪所有无人机已覆盖的边
covered_edges = set()

# 根据 d 将搜索区域划分为 2d x 2d 大小的正方形栅格，并初始化栅格地图
grid_size = 2 * d
num_rows = search_area_y // grid_size
num_cols = search_area_x // grid_size

# 栅格地图初始化
grid_map = np.zeros((num_rows, num_cols))

# 绘制网格线
def draw_grid(ax, search_area_x, search_area_y, grid_size):
    for x in range(0, search_area_x+1, grid_size):
        ax.axvline(x, color='k', linestyle='--', linewidth=0.5)
    for y in range(0, search_area_y+1, grid_size):
        ax.axhline(y, color='k', linestyle='--', linewidth=0.5)

# 定义栅格中心点
def grid_center(row, col, grid_size):
    return (col * grid_size + grid_size // 2, row * grid_size + grid_size // 2)
```

```
def init_obstacles(grid_map, num_obstacles):
    for _ in range(num_obstacles):
        r = random.randint(0, num_rows - 1)
        c = random.randint(0, num_cols - 1)
        grid_map[r][c] = 1    # 在地图上标出障碍物

# 定义障碍物位置并为绘制飞行路径函数添加功能以显示障碍物位置
def draw_obstacles(ax, grid_map, grid_size):
    for row in range(grid_map.shape[0]):
        for col in range(grid_map.shape[1]):
            if grid_map[row][col] == 1:    # 障碍物用 1 表示
                ax.add_patch(Rectangle(
                    (col*grid_size, row*grid_size),
                    grid_size, grid_size,
                    color='grey'
                ))

# 初始障碍物
init_obstacles(grid_map, 80)    # 假设有 20 个障碍物

fig, ax = plt.subplots()
ax.set_xlim([0, search_area_x])
ax.set_ylim([0, search_area_y])
draw_grid(ax, search_area_x, search_area_y, grid_size)
draw_obstacles(ax, grid_map, grid_size)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()

# 飞行路径规划和栅格属性定义，初始化无人机的起始点，使得起始点都是栅格的中心点
def init_drone_paths(grid_map, num_drones, grid_size):
    drone_paths = []
    for _ in range(num_drones):
        valid_start = False
        while not valid_start:
            row = random.randint(0, num_rows - 1)
            col = random.randint(0, num_cols - 1)
            if grid_map[row][col] == 0:    # 确保起始点不是障碍物
                valid_start = True
                drone_paths.append([grid_center(row, col, grid_size)])
    return drone_paths


# 获取下一个位置，确保每次只移动一格且是上下左右
def next_position(current_position, direction):
```

```
    if direction == 'up':
        return (current_position[0], current_position[1] + grid_size)
    elif direction == 'down':
        return (current_position[0], current_position[1] - grid_size)
    elif direction == 'left':
        return (current_position[0] - grid_size, current_position[1])
    elif direction == 'right':
        return (current_position[0] + grid_size, current_position[1])

# 更新检查移动是否在搜索区域内的函数，现在也会检查是否遇到障碍物
def is_valid_move(position, search_area_x, search_area_y, grid_map, grid_size):
    x, y = position
    if 0 <= x < search_area_x and 0 <= y < search_area_y:
        col, row = (int(x / grid_size), int(y / grid_size))
        # 为了避开障碍物，确保当前栅格不是障碍物所在栅格
        return grid_map[row][col] != 1  # 障碍物用 1 表示
    return False

#将路径转换为边的集合。
def find_path_edges(path):
    edges = set()
    for i in range(len(path)-1):
        edge = tuple(sorted([path[i], path[i+1]]))  # 将每段路径排序后转换为元组，确保唯一性
        edges.add(edge)
    return edges

#计算所有无人机路径的重复边的数量。
def calculate_redundancy(drone_paths):
    all_edges = []
    for path in drone_paths:
        edges = find_path_edges(path)
        all_edges.extend(edges)
    unique_edges = set(all_edges)
    redundancy = len(all_edges) - len(unique_edges)
    return redundancy / len(all_edges) if all_edges else 0  # 防止除以 0

# 重新定义飞行路径规划函数以考虑障碍物
def plan_flight_path(path, drone_idx, total_time, T, v, grid_map, grid_size):
    directions = ['up', 'down', 'left', 'right']
    current_position = path[-1]
    while total_time < T:
        direction = random.choice(directions)
        next_pos = next_position(current_position, direction)
        edge = tuple(sorted([current_position, next_pos]))
```

```
            # 如果新位置是有效的且没有障碍物，我们将其添加到路径中
            if is_valid_move(next_pos, search_area_x, search_area_y, grid_map, grid_size)and edge not in
covered_edges:
                path.append(next_pos)
                covered_edges.add(edge)  # 添加到全局已覆盖边集中
                current_position = next_pos
                total_time += grid_size / v
            else:
                # 如果无法移动，则尝试其他方向
                directions.remove(direction)
                if not directions:  # 如果无法移动，则结束循环
                    break

    return path


def flight_time(path,total_time,turn_time,velocity,grid_size):
    straight_distance = 0
    for i in range(2, len(path)):
        current_position=path[i]
        prev_position=path[i-2]
        straight_distance += grid_size
        # 转向加 t 秒
        if i > 1 and (current_position[0] != prev_position[0] or current_position[1] != prev_position[1]):
            total_time += t
    total_time += straight_distance / velocity
    return total_time

def calculate_coverage_rate(path, grid_size):
    coverage_area = 0
    total_area = search_area_x * search_area_y
    for i in range(1, len(path)):
        coverage_area += grid_size * grid_size
    coverage_rate = coverage_area / total_area * 100  # 将这个计算移到循环之后
    return coverage_rate

drone_paths = init_drone_paths(grid_map, N, grid_size)

drone_paths = [plan_flight_path(path, i, 0, T, v, grid_map, grid_size) for i, path in enumerate(drone_paths)]

# 对每架无人机执行路径规划
for i in range(N):
    total_time = 0
    plan_flight_path(drone_paths[i], i, total_time, T, v, grid_map, grid_size)
```

```
# 绘制无人机的飞行路径
def draw_flight_paths(ax, drone_paths):
    for path in drone_paths:
        random_color = (random.random(), random.random(), random.random())
        x_coords = [point[0] for point in path]
        y_coords = [point[1] for point in path]
        ax.plot(x_coords, y_coords, color=random_color)  # 表示搜索路径
         # 在起点处绘制一个圆形标记无人机的位置，使用相同的随机颜色
        ax.scatter([x_coords[0]], [y_coords[0]], color=random_color, s=50, label='无人机',
edgecolor='black')

# 输出每架无人机的飞行路径坐标变化
for idx, path in enumerate(drone_paths, start=1):
    path_str = ' -> '.join(f'({x},{y})' for x, y in path)
    print(f'无人机{idx}飞行路径：{path_str}')

# 输出每架无人机行驶完其路径所耗费的时间
for idx, path in enumerate(drone_paths, start=1):
    total_time = flight_time(drone_paths[i],0,t,v,grid_size)# 直线飞行时间加转向时间
    print(f'无人机{idx}行驶时间：{total_time}秒')

# 绘制初始化的无人机路径和障碍物
fig, ax = plt.subplots()
ax.set_xlim([0, search_area_x])
ax.set_ylim([0, search_area_y])
draw_grid(ax, search_area_x, search_area_y, grid_size)
draw_obstacles(ax, grid_map, grid_size)
draw_flight_paths(ax, drone_paths)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()

def simulated_annealing(grid_map, drone_paths, max_time, turn_time, velocity, grid_size, N):
    initial_temp = 1000.0
    final_temp = 0.001
    alpha = 0.99  # 温度下降系数
    current_temp = initial_temp

    # 计算初始解的覆盖率作为当前最优解
    best_coverage = sum(calculate_coverage_rate(path, grid_size) for path in drone_paths)
    best_paths = drone_paths

    while current_temp > final_temp:
        for i in range(N):
```

```
            if max_time <= current_temp * alpha:  # 模拟时间冷却，退出条件
                break
            total_time = flight_time(best_paths[i], 0, turn_time, velocity, grid_size)
            # 生成新的解（对每架无人机的路径进行小幅调整）
            new_paths = adjust_paths(best_paths, grid_size, grid_map)
            new_coverage = sum(calculate_coverage_rate(path, grid_size) for path in new_paths)

            # 计算解的差异
            delta_coverage = new_coverage - best_coverage

            # 接受准则
            if delta_coverage > 0 or random.random() < math.exp(delta_coverage / current_temp):
                best_paths = new_paths
                best_coverage = new_coverage

            # 温度下降
            current_temp *= alpha

    return best_paths, best_coverage


def adjust_paths(drone_paths, grid_size, grid_map):
    global covered_edges
    # 为所有无人机初始化路径集合
    for path in drone_paths:
        edges = find_path_edges(path)
        for edge in edges:
            covered_edges.add(edge)  # 添加初始边到覆盖集合中

    new_paths = []
    for path in drone_paths:
        new_path = path.copy()
        for i in range(1, len(path) - 1):  # 不调整起点和终点
            prev_point = new_path[i - 1]
            current_point = new_path[i]
            next_point = new_path[i + 1]
            adjustments = ['x', 'y']
            random.shuffle(adjustments)

            # 检查调整后的点是否有效，即既不超出边界也不会进入障碍物
            for adjustment in adjustments:
                new_point = None
                if adjustment == 'x':
                    new_x = next_point[0] if prev_point[1] == current_point[1] else prev_point[0]
```

```
                    new_point = (new_x, current_point[1])
                else:
                    new_y = next_point[1] if prev_point[0] == current_point[0] else prev_point[1]
                    new_point = (current_point[0], new_y)

                if new_point is not None and is_valid_move(new_point, search_area_x, search_area_y,
grid_map, grid_size):
                    # 更新路径点
                    new_path[i] = new_point
                    # 更新覆盖的路径集合
                    covered_edges.discard(tuple(sorted([prev_point, current_point])))
                    covered_edges.discard(tuple(sorted([current_point, next_point])))
                    covered_edges.add(tuple(sorted([prev_point, new_point])))
                    covered_edges.add(tuple(sorted([new_point, next_point])))
                    break  # 成功调整后跳出内部循环

        new_paths.append(new_path)

    return new_paths



# 计算并输出每次调整路径后的路径重复率以及路径覆盖率
def print_path_metrics(drone_paths):
    redundancy_rate = calculate_redundancy(drone_paths)
    print(f"路径重复率: {redundancy_rate*100:.2f}%")
    # 假设 coverage_rate 函数已实现，计算所有无人机的路径覆盖率之和
    total_coverage_rate = sum(calculate_coverage_rate(path, grid_size) for path in drone_paths)
    print(f"综合路径覆盖率: {total_coverage_rate - redundancy_rate*100:.2f}%")


# 使用模拟退火算法对无人机的路径进行优化
best_paths, best_coverage = simulated_annealing(grid_map, drone_paths, T, t, v, grid_size, N)

# 输出每架无人机的飞行路径坐标变化
for idx, path in enumerate(best_paths, start=1):
    path_str = ' -> '.join(f'({x},{y})' for x, y in path)
    print(f'无人机{idx}飞行路径：{path_str}')

# 输出每架无人机行驶完其路径所耗费的时间
for idx, path in enumerate(best_paths, start=1):
    total_time = flight_time(best_paths[i],0,t,v,grid_size) # 直线飞行时间加转向时间
    print(f'无人机{idx}行驶时间：{total_time}秒')
```

```
print_path_metrics(best_paths)

# 绘制优化后的无人机路径和障碍物
fig, ax = plt.subplots()
ax.set_xlim([0, search_area_x])
ax.set_ylim([0, search_area_y])
draw_grid(ax, search_area_x, search_area_y, grid_size)
draw_obstacles(ax, grid_map, grid_size)
draw_flight_paths(ax, best_paths)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```

5. 问题二的 Matlab 代码

(1) 实例一主体文件 q2.m

```
%题目相关参数（第一个例子）

M=12;%目标数量

N=2;%无人机数量

T=1000;%搜索时间，单位：秒

v=0.13;%无人机速度，单位；米每秒

start=[0,0];%起点

d=5;%搜索半径，单位：米

maxdistance=v*T;%单个无人机最大的搜索距离，可以通过调整 v，T 来调整

load("example_1_location.mat");%加载提前生成好的坐标以及权重向
量
load("example_1_weight.mat");

%画出目标物离散图

example_1=figure('Name','实例一','NumberTitle','off');
x=location(:,1)';
y=location(:,2)';
```

```
scatter(x,y,weight/2+10*ones(1,M),'red','filled');
for i = 1:M
text(x(i)+2, y(i), num2str(i), 'HorizontalAlignment', 'left',
'VerticalAlignment', 'middle');
end
hold on;
scatter(start(1),start(2),100,'blue','filled');
text(start(1)+5,start(2)+5,'起点','HorizontalAlignment',
'left', 'VerticalAlignment', 'middle')
xlim([-5,110]);
ylim([-5,110]);
title('随机生成的目标物散点图（目标物权重越大，点半径越大）');
xlabel('横坐标，单位：米');
ylabel('纵坐标，单位：米');


position=zeros(M,5);%目标物位置矩阵
position(:,1)=1:M';%第一列为目标点序号
position(:,2:3)=location;%第二三列为横纵坐标
position(:,5)=weight;%第五列为权重
for i=1:M
dx=position(i,2:3)-start;
position(i,4)=(sum(dx.^2))^0.5;%第四列为到起点的距离
end
weightsum=sum(position(:,5));%总权重


%定义模拟退火参数
G=100000;%迭代次数
T_initial=100;%初始温度
cooling_rate=0.9999;%冷却率
```

```
T_final=0.001;%最终温度


%模拟退火算法主体

current_solution =cell(N,1);
%当前路径初始化
for i = 1:N
current_solution{i}=[i];
%初始化路径，这是一种可达到的较为简单的路径
end
current_weight=calculateweight(current_solution,position,
N);
%当前权重和初始化
[current_distance,~]=shortestway(start,current_solution,p
osition,N,d);
%当前路径长初始化

best_solution = current_solution;
%最佳解初始化
best_distance = current_distance;
%最佳和距离初始化
best_weight = calculateweight(best_solution,position,N);
%计算最佳解的总距离

history=[best_weight/weightsum];
%记录每次退火的权重占总权重的比重


%开始退火
for generation = 1:G
new_solution =
perturbSolution(current_solution,start,d,position,M,N,max
distance);
%扰动当前解生成新解
```

```
new_weight = calculateweight(new_solution,position,N);
% 计算新解的权重和
[new_distance,~]=shortestway(start,new_solution,position,
N,d);
%计算新解的路程


% 判定是否接受新解
flag=0;
temperature = T_initial * (cooling_rate^(generation - 1));
% 决定是否接受新解，flag 为 1 则接受新解
if new_weight>best_weight
%新权重大于最优权重，找到更优的路径，接受新解
flag=1;
elseif new_weight==best_weight && sum(new_distance) <
sum(best_distance)
%新权重等于最优权重且新路径更短时，接受新解
flag=1;
elseif new_weight==best_weight &&
rand<exp((sum(best_distance)-sum(new_distance))/temperatu
re)
%新权重等于最优权重，但新路径更长时，有概率接受
flag=1;
elseif new_weight<best_weight &&
rand<exp((new_weight-best_weight)/temperature)
%新权重小于最优权重，有概率接受
flag=1;
end
%判定是否接受新解
if flag==1
best_solution = new_solution;%更新解

best_distance = new_distance;%更新距离和

best_weight = new_weight;%更新最佳解的总权重
```

```
end
%更新当前解
current_solution=new_solution;
current_distance=new_distance;
current_weight=new_weight;
history=[history,best_weight/weightsum];
% 检查是否达到最终温度或迭代结束
if temperature < T_final
break;
end
end

%退火结束，输出最优解

example_2=figure('Name','实例一求解','NumberTitle','off');
x=location(:,1)';
y=location(:,2)';
scatter(x,y,weight/2+10*ones(1,M),'red','filled');
hold on;
scatter(start(1),start(2),100,'blue','filled');
text(start(1)+5,start(2)+5,'起点','HorizontalAlignment',
'left', 'VerticalAlignment', 'middle')
xlim([-5,110]);
ylim([-5,110]);
title('求解得到的无人机规划路径');

xlabel('横坐标，单位：米');

ylabel('纵坐标，单位：米');
[~,best_path]=shortestway(start,best_solution,position,N,
d);
for i = 1:N
hold on;
x=best_path{i}(:,2)';
y=best_path{i}(:,3)';
plot(x,y,'green');
end

%模拟退火算法效率可视化
```

```
example_3=figure('Name','算法效率可视化
','NumberTitle','off');
plot(0:generation,history);
title('模拟退火算法效率');
xlabel('模拟退火次数：G');
ylabel('最优解路径权重占总权重比重');
xlim([0,100]);
ylim([0,1]);
```

⑵ 实例二主体文件 q2_2.m

```
%题目相关参数（第二个例子）
M=200;%目标数量
N=6;%无人机数量
T=1100;%搜索时间，单位：秒
v=1;%无人机速度，单位；米每秒
start=[0,0];%起点
d=5;%搜索半径，单位：米
maxdistance=v*T;%单个无人机最大的搜索距离，可以通过调整 v，T 来调整
load("example_2_location.mat");%加载提前生成好的坐标以及权重向
量
load("example_2_weight.mat");

%画出目标物离散图
example_1=figure('Name','实例二','NumberTitle','off');
x=location(:,1)';
y=location(:,2)';
scatter(x,y,weight/2+10*ones(1,M),'red','filled');
for i = 1:M
```

```
text(x(i)+5, y(i), num2str(i), 'HorizontalAlignment', 'left',
'VerticalAlignment', 'middle');
end
hold on;
scatter(start(1),start(2),100,'blue','filled');
text(start(1)+10,start(2)+10,'起点','HorizontalAlignment',
'left', 'VerticalAlignment', 'middle')
xlim([-550,550]);
ylim([-550,550]);
plot([-500,-500,500,500,-500],[-500,500,500,-500,-500]);
title('随机生成的目标物散点图（目标物权重越大，点半径越大）');
xlabel('横坐标，单位：米');
ylabel('纵坐标，单位：米');


position=zeros(M,5);%目标物位置矩阵
position(:,1)=1:M';%第一列为目标点序号
position(:,2:3)=location;%第二三列为横纵坐标
position(:,5)=weight;%第五列为权重
for i=1:M
dx=position(i,2:3)-start;
position(i,4)=(sum(dx.^2))^0.5;%第四列为到起点的距离
end
weightsum=sum(position(:,5));%总权重


%定义模拟退火参数
G=200000;%迭代次数
T_initial=100;%初始温度
cooling_rate=0.99995;%冷却率
T_final=0.001;%最终温度
```

```
%模拟退火算法主体

current_solution =cell(N,1);
%当前路径初始化
for i = 1:N
current_solution{i}=[i];
%初始化路径，这是一种可达到的较为简单的路径
end
current_weight=calculateweight(current_solution,position,
N);
%当前权重和初始化
[current_distance,~]=shortestway(start,current_solution,p
osition,N,d);
%当前路径长初始化

best_solution = current_solution;
%最佳解初始化
best_distance = current_distance;
%最佳和距离初始化
best_weight = calculateweight(best_solution,position,N);
%计算最佳解的总距离

history=[best_weight/weightsum];
%记录每次退火的权重占总权重的比重



%开始退火
for generation = 1:G
new_solution =
perturbSolution(current_solution,start,d,position,M,N,max
distance);
%扰动当前解生成新解
```

```
new_weight = calculateweight(new_solution,position,N);
% 计算新解的权重和
[new_distance,~]=shortestway(start,new_solution,position,
N,d);
%计算新解的路程


% 判定是否接受新解
flag=0;
temperature = T_initial * (cooling_rate^(generation - 1));
% 决定是否接受新解，flag 为 1 则接受新解
if new_weight>best_weight
%新权重大于最优权重，找到更优的路径，接受新解
flag=1;
elseif new_weight==best_weight && sum(new_distance) <
sum(best_distance)
%新权重等于最优权重且新路径更短时，接受新解
flag=1;
elseif new_weight==best_weight &&
rand<exp((sum(best_distance)-sum(new_distance))/temperatu
re)
%新权重等于最优权重，但新路径更长时，有概率接受
flag=1;
elseif new_weight<best_weight &&
rand<exp((new_weight-best_weight)/temperature)
%新权重小于最优权重，有概率接受
flag=1;
end
%判定是否接受新解
if flag==1
best_solution = new_solution;%更新解

best_distance = new_distance;%更新距离和

best_weight = new_weight;%更新最佳解的总权重
```

```
end
%更新当前解
current_solution=new_solution;
current_distance=new_distance;
current_weight=new_weight;
history=[history,best_weight/weightsum];
% 检查是否达到最终温度或迭代结束
if temperature < T_final
break;
end
end

%退火结束，输出最优解

example_2=figure('Name','实例二求解','NumberTitle','off');
x=location(:,1)';
y=location(:,2)';
scatter(x,y,weight/2+10*ones(1,M),'red','filled');
hold on;
scatter(start(1),start(2),100,'blue','filled');
text(start(1)+25,start(2)+25,'起点','HorizontalAlignment',
'left', 'VerticalAlignment', 'middle')
xlim([-550,550]);
ylim([-550,550]);
plot([-500,-500,500,500,-500],[-500,500,500,-500,-500]);
title('求解得到的无人机规划路径');

xlabel('横坐标，单位：米');

ylabel('纵坐标，单位：米');
[~,best_path]=shortestway(start,best_solution,position,N,
d);
for i = 1:N
hold on;
x=best_path{i}(:,2)';
y=best_path{i}(:,3)';
plot(x,y,'green');
end
```

```
%模拟退火算法效率可视化

example_3=figure('Name','算法效率可视化
','NumberTitle','off');
plot(0:generation,history);
title('模拟退火算法效率');

xlabel('模拟退火次数：G');

ylabel('最优解路径权重占总权重比重');
xlim([0,generation]);
ylim([0,1]);
```

(3) 计算路径总权重函数文件 calculateweight.m

```
function totalweight=calculateweight(solution,position,N)
%输入路径集，目标位置集，得到权重和
totalweight=0;
for i=1:N
totalweight=totalweight+sum(position(solution{i},5));
end
end
```

(4) 扰动产生新解函数文件 perturbSolution.m

```
function new_solution = perturbSolution(current_solution,
start, d, position, M, N, maxdistance)
% 定义扰动类型
perturbationFunctions = {@perturbSwapDronesTarget,
@perturbAddUnvisitedTarget,
@perturbReplaceVisitedWithUnvisited};
numPerturbationTypes = 3; % 设置 3 种不同的扰动方式

currentState = randi(numPerturbationTypes); % 当前尝试的扰动
类型索引
attempt = 0;
% 尝试扰动，最多尝试 3 次
while attempt < numPerturbationTypes;
% 随机选择或按顺序选择一种扰动方法
```

```
new_solution =
perturbationFunctions{currentState}(current_solution,
start, d, position, M, N, maxdistance);
attempt = attempt+1;
% 检查所有无人机的路径长度是否满足条件
allPathsValid = true;
[distance,~] =
shortestway(start,new_solution,position,N,d);
for i = 1:N
if distance(i) > maxdistance
allPathsValid = false;
new_solution = current_solution;
break;
end
end
if allPathsValid
break;
else
% 获取除当前状态外的所有可能状态
availableStates = setdiff(1:numPerturbationTypes,
currentState);
currentState =
availableStates(randi(numel(availableStates)));
end
end
end

function new_solution =
perturbSwapDronesTarget(current_solution, ~, ~, ~, ~, N, ~)
%扰动一：随机选择两架无人机兑换某一目标点
new_solution = current_solution;
drone1 = randi(N); % 随机选择第一个无人机
drone2 = drone1;
while drone2 == drone1
drone2 = randi(N); % 确保选择不同的无人机
end
targetInDrone1 = randi(numel(new_solution{drone1})); % 选择
第一个无人机的一个随机目标
```

```
targetInDrone2 = randi(numel(new_solution{drone2})); % 选择
第二个无人机的一个随机目标

% 交换目标
temp = new_solution{drone1}(targetInDrone1);
new_solution{drone1}(targetInDrone1) =
new_solution{drone2}(targetInDrone2);
new_solution{drone2}(targetInDrone2) = temp;
end

function new_solution =
perturbAddUnvisitedTarget(current_solution, ~, ~, ~, M, N,
~)
%扰动二：随机添加一个未途经的目标点
new_solution = current_solution;
unvisitedTargets = setdiff(1:M,
unique([new_solution{:}])); % 找出未访问的目标

if ~isempty(unvisitedTargets) % 如果有未访问的目标
targetToAdd =
unvisitedTargets(randi(numel(unvisitedTargets))); % 随机选择
一个添加
else
targetToAdd=[];
end
drone= randi(N); % 随机选择一个无人机
new_solution{drone} = [new_solution{drone},targetToAdd];
end

function new_solution =
perturbReplaceVisitedWithUnvisited(current_solution, ~, ~,
~, M, N, ~)
%扰动三：随机替换一个未途经的目标点
new_solution = current_solution;
unvisitedTargets = setdiff(1:M,
unique([new_solution{:}])); % 找出未访问的目标
```

```
if ~isempty(unvisitedTargets) % 如果有未访问的目标
targetToAdd =
unvisitedTargets(randi(numel(unvisitedTargets))); % 随机选择
一个添加
else
targetToAdd=[];
end
drone = randi(N); % 随机选择一个无人机
targetInDrone_ = randi(numel(new_solution{drone}));
new_solution{drone}(targetInDrone_) = targetToAdd;
end

%function new_solution =
perturbDeleteRandom(current_solution, ~, ~, ~, ~, N, ~)
% %扰动四：随机删去一个途经的目标点
% new_solution = current_solution;
% drone = randi(N); % 随机选择一个无人机
% targetInDrone_ = randi(length(new_solution{drone}));
% new_solution{drone}(targetInDrone_)=[];
%end

%%%第四种扰动方式可以不添加
```

(5) 获得路径函数文件 shortestway.m

```
function [totalDistance, path] = shortestway(startPoint,
solution, position, N, radius)
% 最近邻算法寻找较短路径

% startPoint: 起点坐标 [x, y]

% targetPoints: 第一列为目标物序号,第二三列坐标，第四列到起点距离

% radius: 圆形的半径
path=cell(N,1);
totalDistance = [];
for i = 1:N
```

```
targetPoints=position(solution{i},1:4);
targetPoints=sortrows(targetPoints,4);%按照距离由小到大排序
n = size(targetPoints,1); % 目标圆心数量
currentPlace = startPoint; % 初始化当前位置为起点
path{i} = zeros(n+1,3); % 初始化路径记录矩阵，第一行用于存储起点，
第三列用于储存目标物序号
path{i}(1,1) = 0;%默认起点处序号为 0
path{i}(1,2:3) = startPoint; % 存储起点
dis_=0;
% 对每个目标圆心进行循环
for j = 1:n
% 初始化单条路径距离为 0
% 计算当前位置到下一个目标圆心的向量方向
dx = targetPoints(j,2) - currentPlace(1);
dy = targetPoints(j,3) - currentPlace(2);
distanceToCenter = (dx^2 + dy^2)^0.5; % 计算到圆心的实际距离
% 修正到达点为圆的边界上的点，而非圆心
if distanceToCenter > radius
% 当前位置到圆的实际路径是到圆心的距离减去半径（因为是从边界接触）
dis_ = dis_ + (distanceToCenter - radius);
% 计算边界接触点坐标
contactX = targetPoints(j,2) - (dx/distanceToCenter)*radius;
contactY = targetPoints(j,3) - (dy/distanceToCenter)*radius;
else
% 如果已经在圆内，则直接到圆心，这里逻辑可能需根据实际需求调整
contactX = currentPlace(1);
contactY = currentPlace(2);
end
% 更新当前位置为接触点，并记录路径
```

```
currentPlace = [contactX, contactY];
path{i}(j+1,1)=targetPoints(j,1);
path{i}(j+1,2:3) = currentPlace;
end
totalDistance=[totalDistance;dis_];
end
end
```

(6) 生成实例一文件 example1.m

```
M=12;
%目标物数量
location=randi([20,100],[M,2]);
%目标物位置
weight=randi([0,100],[M,1]);
%目标物权重
```

(7) 生成实例二文件 example2.m

```
M=200;
%目标物数量
location=randi([-500,500],[M,2]);
%目标物位置
weight=randi([0,100],[M,1]);
%目标物权重
```