基于多约束的农作物种植策略的运筹优化

摘要

随着农业现代化的发展，农作物种植优化问题是农业管理与经济效益提升中的重要研究课题。本文针对农作物种植规划中的经济效益与外部不确定性因素的问题，基于多目标优化和时间序列模型思想，通过确定作物种植面积、销售价格、种植成本等指标，以最大化经济效益和最小化种植成本为目标建立了多目标规划模型，并使用模拟退火算法对模型进行求解。

针对问题一，为制定两种情况下未来 2024-2030 的农作物最优种植方案，本文采取多条件约束模型，通过对同时对题目材料进行多方面分析，考虑到不同地块类型、亩产量、以及不同作物之间的相互作用，从而构建一个以多维度求解的目标函数。通过对耕种方面的总和考量，建立了种植大豆后增产、不连茬种植、三年内必种一次豆类作物，种植地块面积和特殊作物不能合种，一共五项约束条件。经过多模型比较后，采用模拟退火算法求解出 23 年—30 年最佳种植方案。结果放置在附件 3 中 result1_1h 和 result1_2。

针对问题二，基于问题一的基础上，问题二将不同作物预期销售量变化趋势不同、农作物亩产量波动、市场对耕种成本的影响和农作物价格趋势等各种不确定因素纳入考量范围。根据题意，在第一问约束条件的基础上加入对不确定性因素的考量，最终仍然采用模拟退火算法，得出 2024 年—2030 年最佳农作物种植方案。结果放置在附件三中的 result2。

针对问题三，基于第一问和第二问的基础上加入考虑农作物之间的可替代性、互补性和预期销售量与销售价格、种植成本之间也存在一定的相关性。在第一、二问的基础上建立可替代性、互补性和相关性模型，并形成对应的约束条件，得到最新的多条件约束模型，得出最优种植方案与问题 2 的结果作比较分析。得出在加入农作物之间的可替代性、互补性和销售数据的相关性后得到的方案更加准确，拟合度更好。

关键词：模拟退火算法　时间序列模型　随机扰动处理　多目标规划模型　　不确定习惯优化

一、 问题重述

1.1 问题背景

2022 年 12 月 24 日，习近平总书记在中央农村工作会议上强调，中国要强，农业必须强。自古以来，农业问题始终是头等大事，是满足人民高质量生活需求不可替代的基础条件，是建设农业强国不可或缺的保障。聚焦耕地和种子这两个关键环节，持续提升粮食的综合生产能力，是助力全体人民共同富裕的重要途径，也是实现粮食增产、农户增收和农业增效协同发展的关键，从而推进农业强国建设[1] 。而这些离不开粮食生产所必需的条件——耕地[2]。因此充分利用有限的耕地资源，对乡村经济可持续发展具有重要意义。

某华北山区村落现有 34 个大小不同的地块组成露天耕地共 1201 亩，包括平旱地、梯田、山坡地和水浇地。其中水浇地适宜每年种植一季水稻或两季蔬菜，其余类型的耕地均只能每年种植一季粮食类作物。该乡村另有大棚，普通大棚和智慧大棚分别有 16 个和 4 个，每个耕地面积为 0.6 亩。普通大棚每年种植一季蔬菜和一季食用菌为佳，智慧大棚每年种植两季蔬菜为佳。同一块耕地（含大棚）每季可以和种不同的作物。

根据农作物生长规律，每种作物不能连续重茬种植并且含有豆类作物根瘤菌的耕地更有利于其他作物生长，每个地块的所有土地需在三年内种植一次豆类作物。同时，种植方案应考虑到耕种作业和田间管理的便利。

附件 1 给出乡村现有耕地和农作物的基本情况

附件 2 给出 2023 年乡村农作物种植和相关统计数据

1.2 、问题提出

问题一：假定每季农作物都在当季销售，且农作物未来的**预期销售量**、**种植成本**、**亩产量**和**销售价格**相对 2023 年**保持稳定**。若某作物每季总产量**超过预期销售量**则会无法正常销售，造成浪费。请针对**“超过部分滞销，造成浪费”**和**“超过部分按 2023 年销售价格的 50%降价出售这两种情况”**，给出 2024—2030 年农作物的**最佳种植方案**。

问题二：根据往年经验，小麦和玉米的预期销售量呈**每年增长率** 5%—10%之间的趋势，其他农作物未来**每年销售量预期相对于 2023 年**约有±5%的浮动。农作物亩产量有±**10%**的变动，因气候等因素影响。受市场条件变化，农作物的种植成本每年将增长**5%**左右。**蔬菜类作物**的销售价格每年平均增长 **5%**左右；粮食类作物价格**稳定**。食用菌价格每年**平稳下降约 1%—5%**左右，尤其是羊肚菌每年价格下降约 **5%**。

综合考虑农作物**种植和销售**的各方面条件，设计该乡村 2024—2030 年农作物种植**最优方案**。

问题三：在现实生活中，农作物之间存在一定的**可替代性**和**互补性**。同时，**预期销售量**与**销售价格**、**种植成本**之间也存在一定的**相关性**。在问题 2 的基础上综合考虑，诶出 2024 年—2030 年农作物的**最优种植策略**、求解，并与问题 2 的结果**比较分析**。

二、 问题分析

农作物的种植策略问题给出了乡村现有耕地和农作物的基本情况和 2023 年乡村农作物种植和相关统计数据，题目要求据此制定 23 年之后未来 7 年的种植方案。耕地利用是指人类根据土地的可耕性能，通过一定的耕作和植物栽培行为来利用土地，以满足自身需要的过程。耕地利用的类型会受自然条件、社会条件、经济条件和技术条件等多重因素改变。[3] 最佳种植方案的制定是村民收益、农作物种条件和地块分配的综合决策。本文的最佳种植方案的制定在保证因地制宜，乡村经济的可持续发展的前提下以村民收益最大化为目标进行种植方案的优化。

问题一：问题一要求在两种不同的情况下基于 23 年的种植情况和耕地农作物情况进行优化处理，探寻未来的最优种植方案。首先对数据进行预处理，用 Python 将数据拆分，使所有数据根据编号排列并提取不同作物在不同土地上的产量和不同季节的亩产，进行分析，对作物适宜耕种的季度进行评价，找出作物最适宜种植的季度。

（1）情况一：超过部分滞销，造成浪费。

情况一要求在超过部分滞销造成浪费的前提下制定未来的最佳种植方案。因此，面对乡村现有的耕地和农作物情况，考虑到优先保障收益最大化，有必要对耕地、农作物情况和23年相关种植数据进行分类编号并系统处理，找出各类作物最适宜种植的方案，确保农作物种植方案最优。建立多条件约束模型，对种植方案进行优化，得 2024 年—2030 年的最优种植方案。

（2）情况二：超过部分按 2023 年销售价格的 50%降价出售。

情况二要求在超过部分按 23 年价格的 50%出售的基础上制定未来最佳种植方案。因此，考虑基于情况一所构建的模型，通过按照提议对目标函数和约束条件的修改，得出 24 年—30 年新的最优种植方案。

问题二：问题二要求基于 2023 年农作物种植情况和种植地块相关信息，结合未来 7 年内农作物的销售量、亩产量、种植成本以及价格等不确定性因素，以最大化未来的农作物总收益为目标，制定出最佳的种植方案。首先对附件 2 中的两张表进行分类合并，

用 excel 将 23 年不同地块种植的各种作物与其亩产量、种植成本、销售单价相对应。通过 python 编程实现数据处理及模拟，使用随机扰动方法模拟外部不确定性因素的影响，对不同农作物的种植效益进行多维度分析，探寻每年不同作物在各地块的最佳种植面积。

问题三：问题三要求在考虑作物间的可替代性与互补性基础上，优化未来 7 年内的农作物种植方案，以最大化农作物总收益为目标。可替代性主要体现小麦与玉米之间的可替代性（玉米价格上涨而小麦价格相对较低，模型会倾向于在某些地块上优先种植玉米）。互补性主要体现在豆类作物与粮食作物以及大棚蔬菜与食用菌的轮作。利用问题二合并过后的数据，通过 python 编程处理不同作物之间的替代性与互补性关系。利用替可代性根据市场价格和生产成本灵活调整作物的种植比例，而互补性则通过分析作物间轮作或套种的协同效应。建立多约束函数，优化整体收益。

三、 模型假设与符号说明

3.1 模型基本假设

(1) 假设 2023 年耕地的种植量等于销售量，且无其他因素影响销售量。

(2) 假设农产品的销售价格只会按照理想的状态增减，排除商户之间恶意竞争造成的价格影响。

(3) 假设一块耕地上有多种作物合种时，由于种植区域不一样从而使得农作物在下一年种植时仍然可以种植在该地块的不同区域。

(4) 假设优化过后出现用地不完全的情况，村民会定期大理和护理耕地，不会让其荒废。

3.2 符号说明

表1 符号说明

符号	含义	单位
S	总收益	元
Q	总销售额	元
C	总成本	元
i	地块	/
j	作物	/
k	年份	/
x_{ijk}	第k年第i块地种植的作物j	/
M_{ij}	第i块地作物j的亩产量	/
a_{ij}	第i块地种植的j作物的售价	/

C_{ij}	第i块地种植的j作物的种植成本	/
y_{ijk}	第k年第i地块是否种植的作物	/
m_i	第i块地的面积	/
r_{jn}	物j被作物n替代是的收益调整系数	/
s_{jn}	作物j和作物n的互补性效果	/
$\triangle a_{nk}$	作物n在k年价格的波动比例	/

四、 数据预处理

4.1 数据分析

根据附件 1、附件 2 对列表进行分析，从中发现农作物的生产种植有季节和种植的地块类型之分，亩产量与种植地块之间也拥有这紧密的联系。主要可以分为以下两大类：

单季作物：在农作物当中拥有相当大一部分数据属于单季作物，其特征为一年 1 熟，或者只能在特定的季节环境下才能生长。

双季节作物：该类作物可在特定的种植地块所创造的宜生环境中生长。只占据所有作物中的一小部分。

4.2 数据清洗

使用 **pandas** 读取附件 1、附件 2。提取出其中的作物编号、种植地块、种植季次等参数够建符合附件 3 要求格式的矩阵。并将有关约束条件的参数提取构建成特定的矩阵，使得数据更有利于约束函数的计算。基于问题背景可知，农作物的销售价格趋于稳定，遂对农作物的售价去平均值。

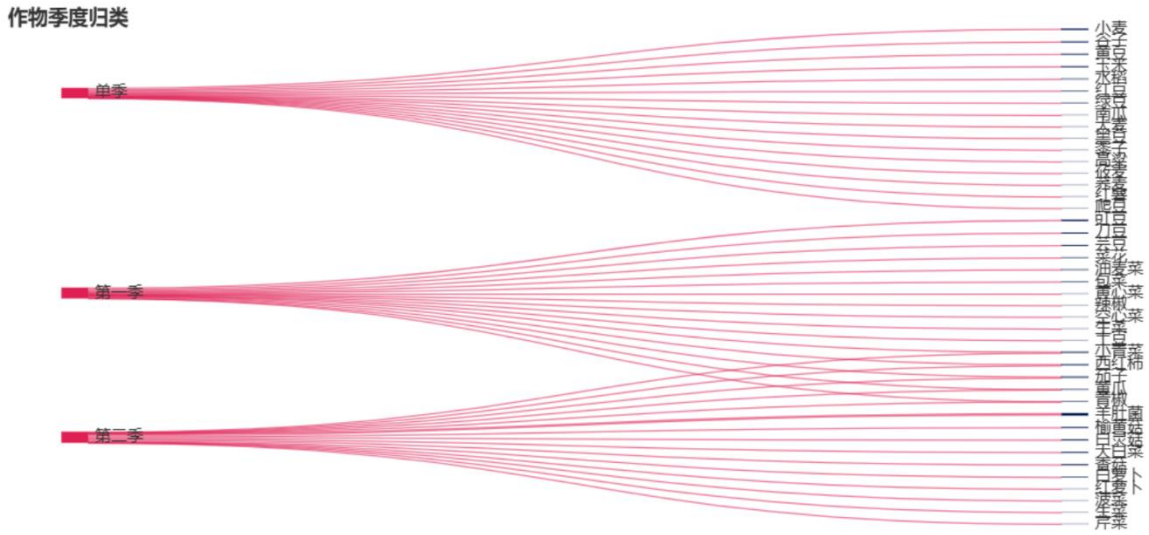

图1 作物归类图

通过对作物耕作时间的归类，显示出第一季度耕种作物与第二季度耕种作物有一定的重合，体现有些作物可以两个季度都种植，某些农作物必须严格按照季度耕作。

五、 问题一模型建立与求解

5.1 问题一求解思路

针对情况一，题目要求超过预期销售量的部分将造成浪费，制定最佳种植方案时需要尽可能减少浪费或者不造成浪费，以最接近预期销售量的种植产能尽可能的减少土地种植成本，通过合理的土地分配最大化种植产能并满足农田的可持续发展。同时对题目材料进行多方面分析，考虑到不同地块类型、亩产量、以及不同作物之间的相互作用，从而构建一个以多维度求解的目标函数。对于材料中的多方面约束，同时构建多维度约束函数对最终目标进行逼近求解。

针对情况二，题目要求超过预期销售量的部分按照 23 年价格的 50%出售，考虑基于情况一所构建的模型，尝试模型和目标函数进行修改，利用多维度约束函数求解得出未来 7 年农作物最优种植方案。

5.2 建模前准备

根据参考文献可知，多条件约束模型通过约束条件优化资源分配，可以提高资源的使用效率。每一项约束条件的建立均为从不同方向反映种植规划时需要考虑到的问题。在多约束模型的构建中，构建的原则在于：目标明确、完整性、一致性、简洁性和可量化。通过查找资料和根据日常生活经验得出恰当的约束条件。

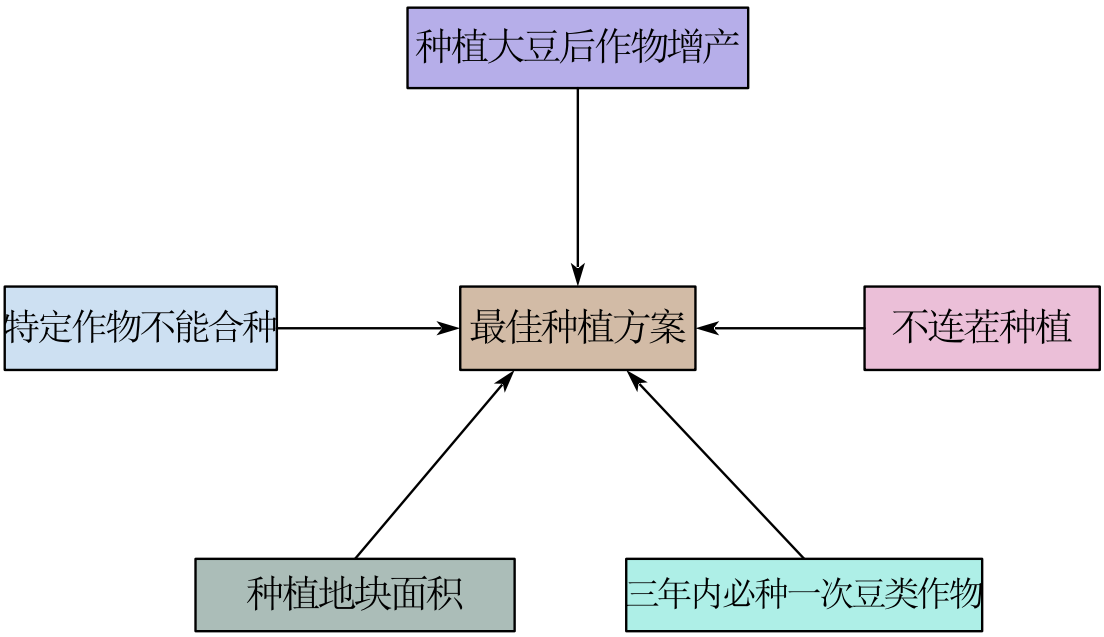

图2 最佳种植方案的多约束条件

5.2.1 种植大豆后作物增产

大豆对土壤氮素有亏损，有增加，也有收支平衡。我们认为大豆肥田机制是同样的。但对土壤氮素含量的盈亏是随土壤气候条件而异，人为的栽培技术更直接影响了大豆根系生长及代谢[4]，因此含有豆类作物根菌的土壤并不能使所有物增产。根据研究，大豆轮作二年后的大豆茬春小麦、玉米和谷子的产量比各自的重茬产量分别提高 59%、16%和 20%[4]。在本文中仅考虑在大豆茬后种植小麦、玉米和谷子的增产情况，其余情况均不考虑。

5.2.2 特定作物不能合种

通过在国内权威种植网站“耕种帮种植网”对农业知识的查询，我们学习了一些与农业有关的知识，了解到有些农作物不能在一起耕种。如果在一起耕种会导致农作物口味变化，比如苦瓜和丝瓜合种会导致丝瓜口感变苦；甚至会导致农作物互相印象对方的生长情况。针对本题附件中给出的农作物，黄瓜和西红柿不能种植在同一地块，因为二者分泌物具有互相抑制对方生长的作用，另外，他们都极易受蚜虫的危害，一旦一方发生蚜虫，另一方很快也会遭受侵袭导致村名造成巨大的损失。

根据本题的情况，本文中不会出现黄瓜和西红柿同时种在同一地块的情况。

5.3 问题一模型的建立

5.3.1 目标函数的建立

依据题意制定未来农作物最优种植方案，最优种植方案的制定需要在考虑乡村经济可持续发展的情况下实现村民收益最大化：

$$\max S = Q - C \tag{1}$$

其中，S即为总收益，Q为总销售额，C为总成本。依据题目中给出的条件，可知：

$$Q = x_{ijk} \times M_{ij} \times a_{ij} \tag{2}$$

$$C = x_{ijk} \times C_{ij} \times M_{ij} \tag{3}$$

其中，x_{ijk}为第k年第i块地种植的作物j，M_{ij}为第i块地作物j的亩产量，a_{ij}为第i块地种植的j作物的售价，C_{ij}为第i块地种植的j作物的种植成本。

联立公式(1)(2)(3)可得出最终的目标函数：

$$\max S = x_{ijk} \times M \times a_{ij} - x_{ijk} \times C_{ij} \times M \tag{4}$$

5.3.2 建立多条件约束模型

通过约束条件优化资源分配，可以提高资源的使用效率，从不同角度反映规划时需要考虑到的问题，从而得出最优农作物种植方案。

a.不能连续重茬种植

根据农作物的生长规律，每种作物在同一地块连续重茬种植会导致减产的后果，据此设置同一种作物在同一地块每两年内不能连续种植的约束函数：

$$\begin{gathered} y_{ijk} \cdot y_{ijk+1} = 0 \\ y_{ijk} \epsilon \{0, 1\} \\ k = 2024, 2025, ..., 2030 \end{gathered} \tag{5}$$

其中，y_{ijk}和y_{ijk+1}为逻辑变量y_{ijk}即为第k年第i地块是否种植的作物j，y_{ijk+1}同理。“1”表示第k年第i地块种植了作物j，“0” 第k年第i地块没有种植作物j。

b.三年内至少种植一次豆类作物

含有豆类作物的根菌的土壤有利于其他作物的生长，从 2023 年开始每个地块的所有土地三年内至少种一次豆类植物：

$$\sum_{k=2024}^{k+3} x_{ijk} \geqslant 1 \tag{6}$$

c.豆类作物根菌土壤促进作物增产

小麦增产：

$$x_{ijk} \cdot Mij \cdot (1+59\%) \tag{7}$$

玉米增产：

$$x_{ij} \cdot M_{ij} \cdot (1+16\%) \tag{8}$$

谷子增产：

$$x_{ij} \cdot M_{ij} \cdot (1+20\%) \tag{9}$$

d.种植地块面积

(1)总种植面积不超过耕地总面积

$$\sum_{k=2023}^{m} x_{ijk} \leqslant m_i \tag{10}$$

其中，m_i表示第i块地的面积。

(2)农作物种植面积不宜太小

为方便农田间管理以及种植的多样化，农作物种植面积不宜过小：

$$\begin{aligned} m_{\text{露天}} &\geqslant 1\text{亩} \\ m_{\text{大棚}} &\geqslant 0.2\text{亩} \end{aligned} \tag{11}$$

其中，$m_{\text{露天}}$即为露天耕地的种植面积，$m_{\text{大棚}}$为大棚内的种植面积。

5.4 问题一模型求解与分析

5.4.1 算法求解

a.模拟退火算法求解流程如下：

模拟退火算法求解流程

Step1 将数据导入 Python，确定目标函数、决策变量和约束条件

Step2 初始化参数

Step3 计算新解的目标函数值与当前解的目标函数值的差值

Step4 如果新解优于当前解，接受新解作为当前解；否则则有一定概率接受新解

Step5 根据降温策略更新温度

Output 2024 年—2030 年最优种植方案

5.4.2 求解结果展示

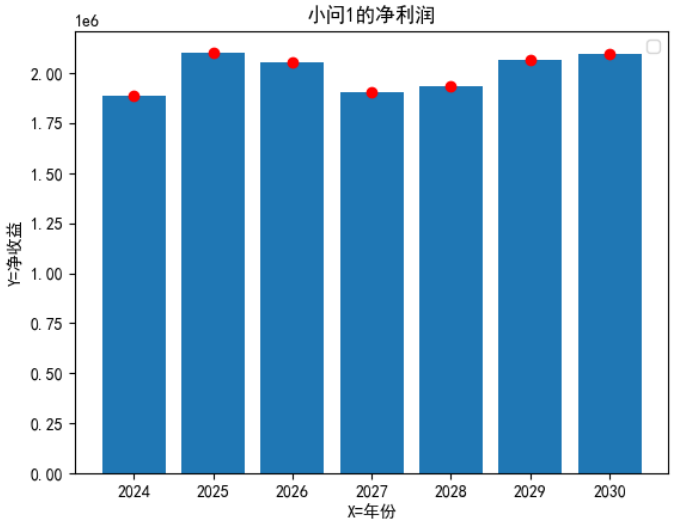

图3 第一小问净利润

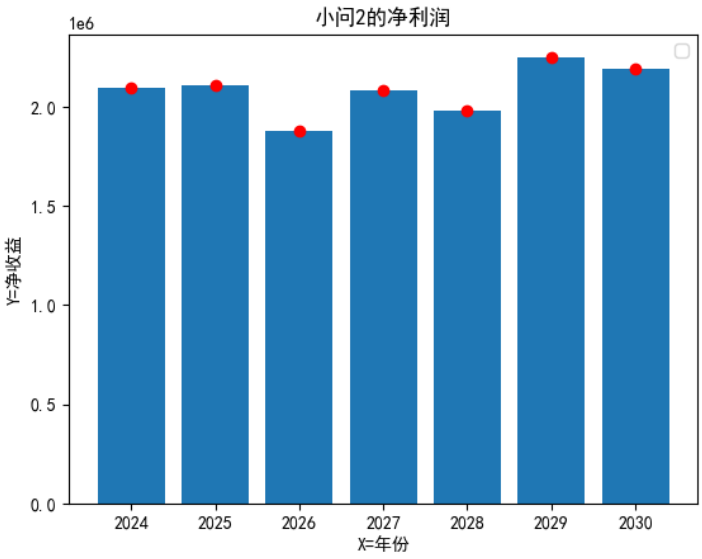

图4 第二小问净利润

通过提润表柱形图可以看出优化过后，收益较为稳定，可以让村民安心在村落发展而不是外出务工。

由于得到的种植方案涉及数据和表格较多，第一问一二小问的答案详见结果放置在附件 3 中 result1_1h 和 result1_2。

5.4.3 求解结果对比

在求解问题的第一小问的过程中，发现在耕地全部利用的情况下优化种植方仍会造成超售滞销。经过思考之后，我们想探究如果在不造成浪费的情况下，尽可能的优化土地种植面积，进而节省耕种成本，从而得到我们收益最大化的目标。因此，我们额外探究了在种植量刚好等于预期销售量时可以获得的最大收益，并且将每年得到的收益汇总制成图表反应探究结果。

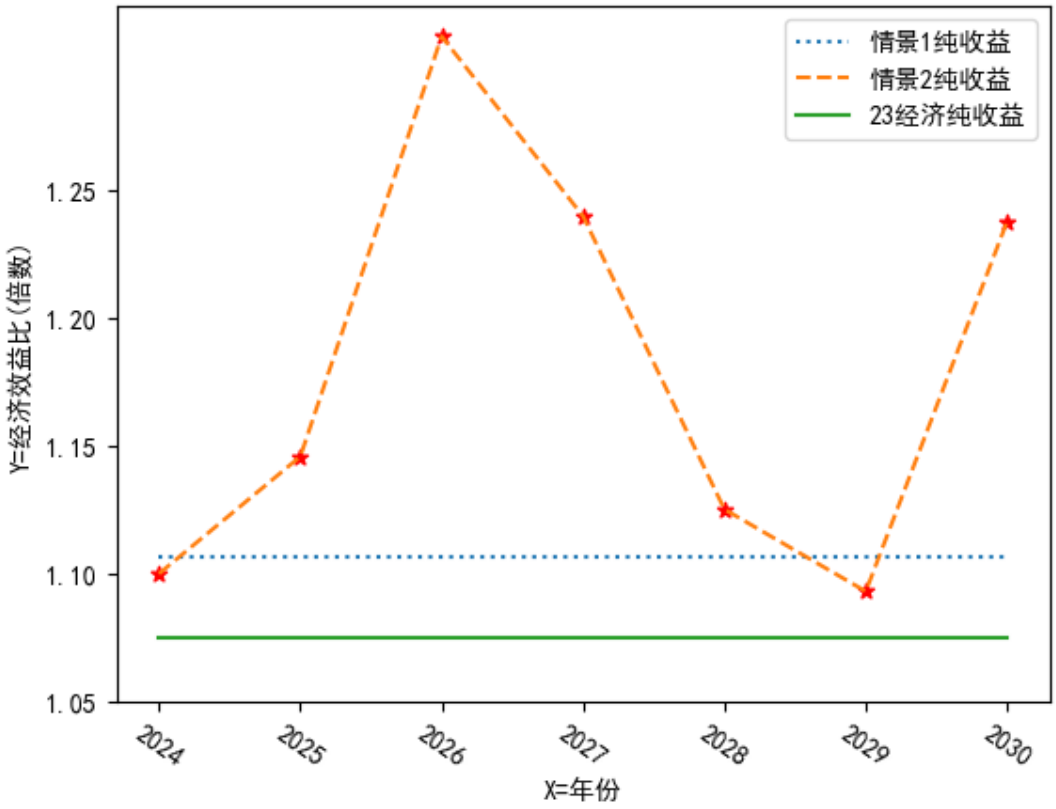

图5 不同思路得到的每年收益汇总折线图

情景一为每年种植量刚好等于预期销售量，优化土地种植面积的情况，情景二为在充分利用土地的情况下，造成浪费的情况。通过与 23 年经济收益对比，得出情景一优化后能稳定保持高出 23 年的收益，几乎没有波动；情景二虽然波动较大，有两年收益低于情况一，但情景二的平均收益远高于情景一。由此可得，情景二得出的结果更加符

合题意，也更能达到经济收益最大化的目标。因此在充分利用土地的情况下造成浪费的方案最佳。

通过对折线图的观察，可以发现情景二的经济收益呈周期性变化，总体上以 6 年一周期的变化进行，2026 年时经济收益达到最高，随后减少，在 2030 年时经济效益又开始上升。导致周期性变化的原因可能是每三年需种一次豆类植物导致作物增产，从而使经济效益升高。

六、 问题二模型建立与求解

6.1 问题二求解思路

基于 2023 年的农作物种植情况，通过随机扰动模拟未来不确定性因素作为约束函数，与题意作物季节性种植限制要求结合成多约束函数，建立最大化农作物的总收益的目标函数。采用模拟退火算法优求解未来 7 年内的种植方案。

6.2 问题二模型建立

6.2.1 销售量波动建模

a.小麦和玉米的销售增长量

小麦和玉米的年增长率介于 5%到 10%之间。现使用一个随机变量g_{wc}表示小麦和玉米的年增长率，可以从中抽取增长率$g_{wc}\in[0.05,0.10]$：

$$g_{wc}\sim Uniform(0.05,0.10) \tag{11}$$

设 2023 年的基础需求量为$D_{wc}(2023)$，则未来k年的需求量为：

$$D_{wt}(k)=D_{wc}(2023)\cdot(1+g_{wc})^{(k-2023)} \tag{12}$$

b.其他农作物销售量波动

其他农作物未来每年的预期销售量相对于 2023 年有大约$\pm5\%$的变化，波动区间取自[-0.05,0.05]:

$$g_o(k)\sim Uniform(-0.05,0.05) \tag{13}$$

其中，$g_o(k)$为年波动。由此可得某一年k的需求量为：

$$D_o(k)=D_o(2023)\cdot(1+g_o(k)) \tag{14}$$

其中，$D_o(2023)$为 2023 年的基础销售量；$D_o(t)$为k年的需求量。

6.2.2 农作物的亩产量建模

受气候等因素影响，农作物亩产量往往会有±10%左右的变化，现有一个随机变量$z(k)$表示年亩产波动，取自区间$[-0.10,0.10]$：

$$z(k)\sim Uniform(-0.10,0.10) \tag{15}$$

则可知，某年k的亩产量$Z_c(k)$：

$$Z_c(k)=Z_c(2023)\cdot(1+z(k)) \tag{16}$$

其中，$Z_c(2023)$为2023年的基础亩产量。

6.2.3 农作物的种植成本建模

受市场条件影响，农作物的种植成本每年平均增长5%。假设2023年的种植成本为$C(2023)$，则某k年的种植成本为：

$$C(k)=C(2023)\cdot(1+0.05)^{(k-2023)} \tag{17}$$

6.2.4 农作物销售价格建模

a.粮食作物的销售价格

粮食作物的销售价格基本稳定，因此粮食类作物在未来年份的销售价格与2023年持平：

$$\begin{gathered} a_j(k)=a_j(2023) \\ j\in 粮食作物 \end{gathered} \tag{18}$$

其中，$a_j(k)$为k年的j粮食作物售价；$a_j(2023)$为2023年j粮食作物售价。

b.蔬菜类作物的销售价格

蔬菜类作物的销售价格每年增长月 5%，所以某一年k的销售价格$a_{ij}(k)$可表示为：

$$\begin{gathered} a_j(k)=a_j(2023)\cdot(1+0.05)^{(k-2023)} \\ j\in 蔬菜 \end{gathered} \tag{19}$$

其中，$a_j(k)$为k年的j蔬菜作物售价；$a_j(2023)$为2023年j蔬菜作物售价。

c.食用菌（羊肚菌除外）的销售价格

食用菌的销售价格稳中有降，大约每年可下降 1%—5%，使用随机变量$d_{mush}\in[0.01,0.05]$表示每年的下降率：

$$d_{mush}\sim Uniform(0.01,0.05) \tag{20}$$

由此可知，某年k的销售价格$a_j(k)$可表示为：

$$a_i(k)=a_i(2023)\cdot(1-d_{mush})^{(k-2023)}$$
$$j\in 菌类 \tag{21}$$

其中，$a_j(k)$为k年的j菌类（除羊肚菌外）作物售价；$a_j(2023)$为2023年菌类（除羊肚菌外）作物售价。

d.羊肚菌销售价格

羊肚菌销售价格每年下降幅度为5%：

$$a_{羊}(k)=a_{羊}(2023)\cdot(1-5\%)^{(k-2023)} \tag{22}$$

其中，$a_j(k)$为k年的羊肚菌售价；$a_j(2023)$为2023年羊肚菌售价。

6.2.5 多条件约束模型的补充建立

在基于问题一模型的基础上根据问题二的题目要求修改、添加约束条件。

a.季节性约束

(1)**平旱地、梯田、山坡地：**只能种植单机粮食类作物（水稻除外）

$$x_{ijk}=0,若作物j不是粮食类作物（排除水稻） \tag{23}$$

(2)**水浇地：**可以单季种水稻，或者种两季蔬菜作物。若第一季种蔬菜(排除大白菜、白萝卜、红萝卜)，则第二季必须种植大白菜、白萝卜、红萝卜之一：

$$若\sum_{j\in 蔬菜类}x_{ijk}^{第一季}>0\ 则\sum_{j\in\{大白菜，白萝卜，红萝卜\}}x_{ijk}^{第二季}>0 \tag{24}$$

(3)**普通大棚：**每年种两季作物，第一季可以种蔬菜类作物，第二季需要种植食用菌。

$$\sum_{j\in 蔬菜类}x_{ijk}^{第一季}>0\ 且\sum_{j\in 食用菌类}x_{ijk}^{第二季}>0 \tag{25}$$

(4)**智慧大棚：**可以种植两季蔬菜，但是第一季不能种大白菜、白萝卜、红萝卜。

$$x_{ijk}=0,若j\in\{大白菜，白萝卜，红萝卜\}\ 且第一季 \tag{26}$$

6.3 问题二模型求解与分析

利用模拟退火算法基于问题一求解出2024年—2030年的最佳种植方案，根据求解出的数据制作图标：

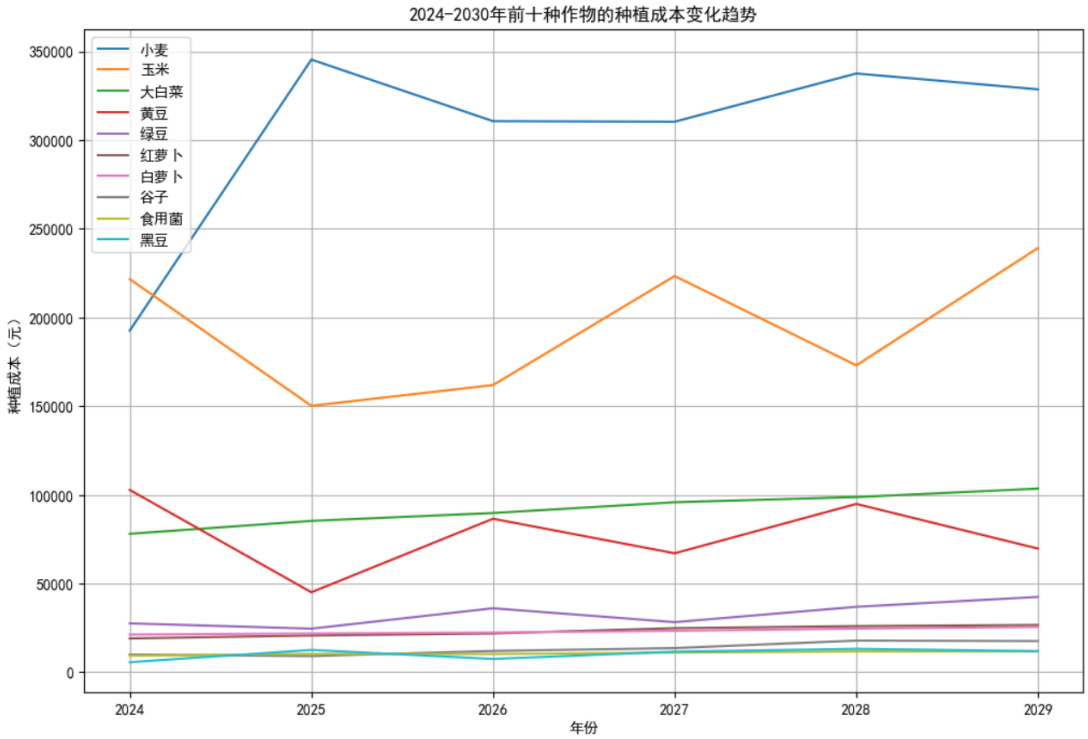

图6 2024-2030 年前十种作物的种植成本变化趋势

由折现图可知，小麦的种植成本在 24 年到 25 年突然增加，之后基本稳定；其余作物的成本虽有波折，但长远来看比较平稳。小麦的成本除 24 年外一直居高不下，但小麦的种植成本较低，因此表明该村庄小麦销量较好，大部分村民依靠种植小麦为生。

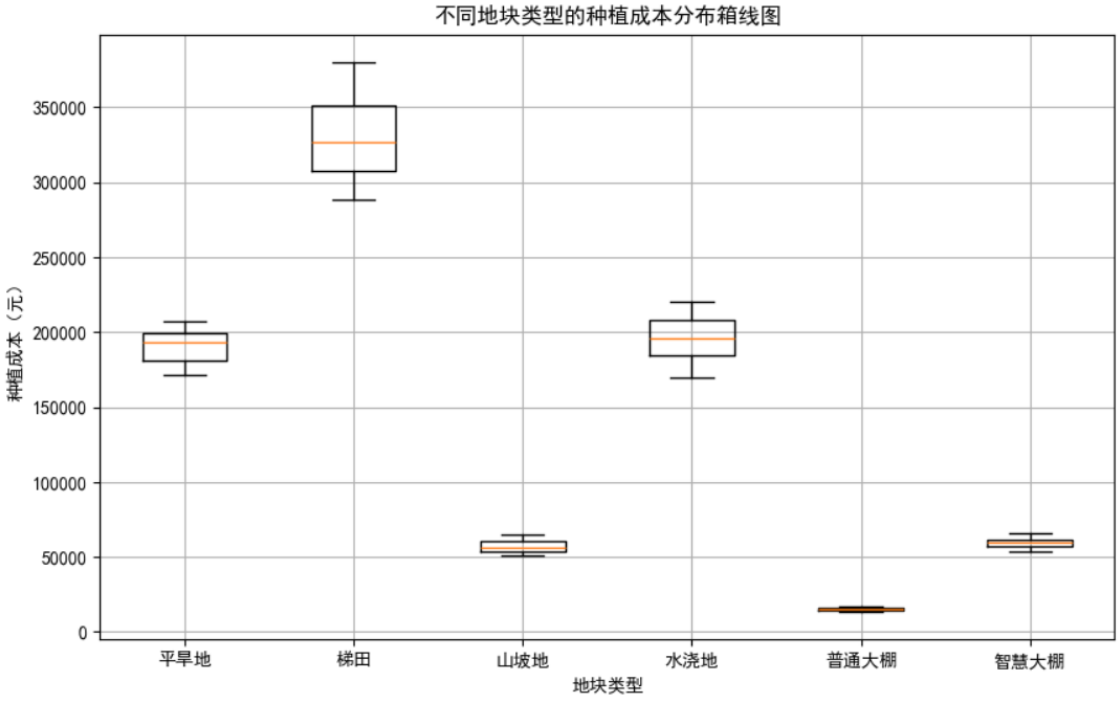

图7 不同地块类型的种植成本分布箱线图

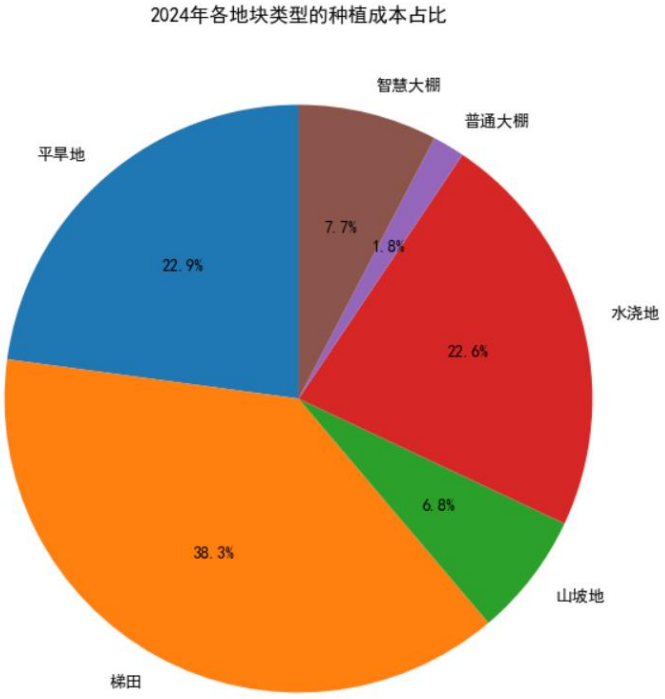

图8 2024 年各地块类型的种植成本占比

通过各地块类型种植成本箱线图和饼图的对比，可直观看到梯田的种植成本最高，其次是平旱地。当村名面对着两样成本高昂的地块时，需要优先考虑种植利润率较高的作物。

具体方案数据答案详情见附录 3result2。

七、 问题三模型建立与求解

7.1 问题三求解思路

在第二问的基础上，引入相关替代性和互补性约束，建立作物价格/成本相关性矩阵，结合未来的市场波动情况和种植成本等不确定因素，构建多目标函数以最大化农作物的总收益。采用模拟退火算法对这些多目标和约束条件进行求解，优化未来 7 年内的最佳种植方案。

7.2 问题三模型建立

7.2.1 模型的建立

a.可替代性模型

农作物的可替代性：在农业经济学中，农作物的可替代性是指一种农作物能够被另一种农作物所替代，以满足市场上消费者的需求或生产者的生产目标。这种替代性通常涉及到作物的功能性、营养价值、口感、价格和可获得性等方面。当一种作物的供应减少或价格上升时，消费者可能会寻找并转向其他具有相似特性的作物，从而影响市场对这些作物的需求。

根据题目中给出的农作物信息并结合查阅到的资料，可以大致按农作物的分类提档线农作物的可替代性，同类型的农作物之间可以相互替代。

设定替代系数r_{jn}，表示作物j被作物n替代是的收益调整系数（通常 0<r_{jn} ⩽1）。当满足一定条件时，作物j可以被作物n替代。

当$\frac{a_{jk}}{C_{jk}}$比$\frac{a_{nk}}{C_{nk}}$小，且替代作物n的市场表现更好时，作物j被作物n替代：

$$\begin{gathered} x_{ijk}=0,\ x_{ink}=S_{ink} \\ \frac{a_{nk}}{C_{nk}}>\frac{a_{jk}}{C_{jk}},\ n\in \text{替代作物} \end{gathered} \tag{27}$$

b.互补性模型

农作物的互补性：在农业经济学中，农作物的可替代性是指一种农作物能够被另一种农作物所替代，以满足市场上消费者的需求或生产者的生产目标。这种替代性通常涉及到作物的功能性、营养价值、口感、价格和可获得性等方面。当一种作物的供应减少或价格上升时，消费者可能会寻找并转向其他具有相似特性的作物，从而影响市场对这些作物的需求。

根据现实生活中的经验和查阅到的资料，本文挑选了一些在种植方法、营养成分方面互补的农作物作为互补品。

定义互补性系数s_{jn}，表示作物j和作物n的互补性效果。这个系数用于提高后续作物的收益或产量。

如果作物j在某地块钱被种植过，则作物j在该地块的后续种植会带来附加收益：

$$M_{jk}=M_{jk}\cdot(1+s_{jn}) \tag{28}$$

其中$s_{jn}>0$表示作物j和作物n具有互补性。

c.相关性模型

定义一个相关性矩阵R_{jn}，描述作物j和作物n在市场价格或者需求上涨的相关性。比如，玉米和小麦价格具有较强的正相关性，那么当玉米价格上升时，小麦价格也会随之上升。

若作物j和作物n的相关性系数为ρ_{jn}，那么作物j的市场价格受作物n影响：

$$a_{jk} = a_{jk} \cdot (1 + \rho_{jn} \cdot \triangle a_{nk}) \tag{29}$$

其中，$\triangle a_{nk}$是作物n在k年价格的波动比例。

类似的，作物的种植成本也会受到市场条件的相关性影响，种植成本可以通过相关性举证进行调整：

$$C_{jk} = C_{jk} \cdot (1 + \rho_{jn} \cdot \triangle C_{nk}) \tag{30}$$

7.2.2 可替代性约束条件

问题三是基于问题一和问题二的思路和模型，考虑到可替代性，对未来 7 年的最佳种植方案进行再优化。

在作物间具有替代性的情况下，如果某作物的成本过高或价格过低，种植策略会将其替换为更具经济效益的作物。替代作物的选择可以通过以下条件描述:

$$\frac{a_{jk}}{C_{jk}} < \frac{a\,\text{替代作物},k}{C\,\text{替代作物},k} \tag{31}$$

当上述条件成立时，作物j被作替代为物n，即：

$$x_{ijk} = 0,\ x_{ink} > 0 \tag{32}$$

7.2.3 目标函数：最大化总收益

$$\max \sum_{k=2024}^{2030} \sum_{n} \sum_{j} (M_{jk} \cdot a_{ik} \cdot (1 + \rho_{jk}) - C_{jk}) \cdot x_{ijk} \cdot (1 + s_{jn}) \tag{33}$$

7.3 问题三模型求解与分析

结合问题背景，问题二和问题三的主要区别在于是否对种植方案进行了替代性和互补性的优化调整。通过这两个问题的经济效益对比，可以看出优化种植方案对提高总收益和减少波动的显著作用。

7.3.1 求解结果

利用模拟退火算法基于问题一、问题二和新的限制条件优化 2024 年—2030 年的最佳种植方案，因为数据庞大，答案详情见附录 3。下图为 2024 年—2030 年最优种植方案可视化柱状图。

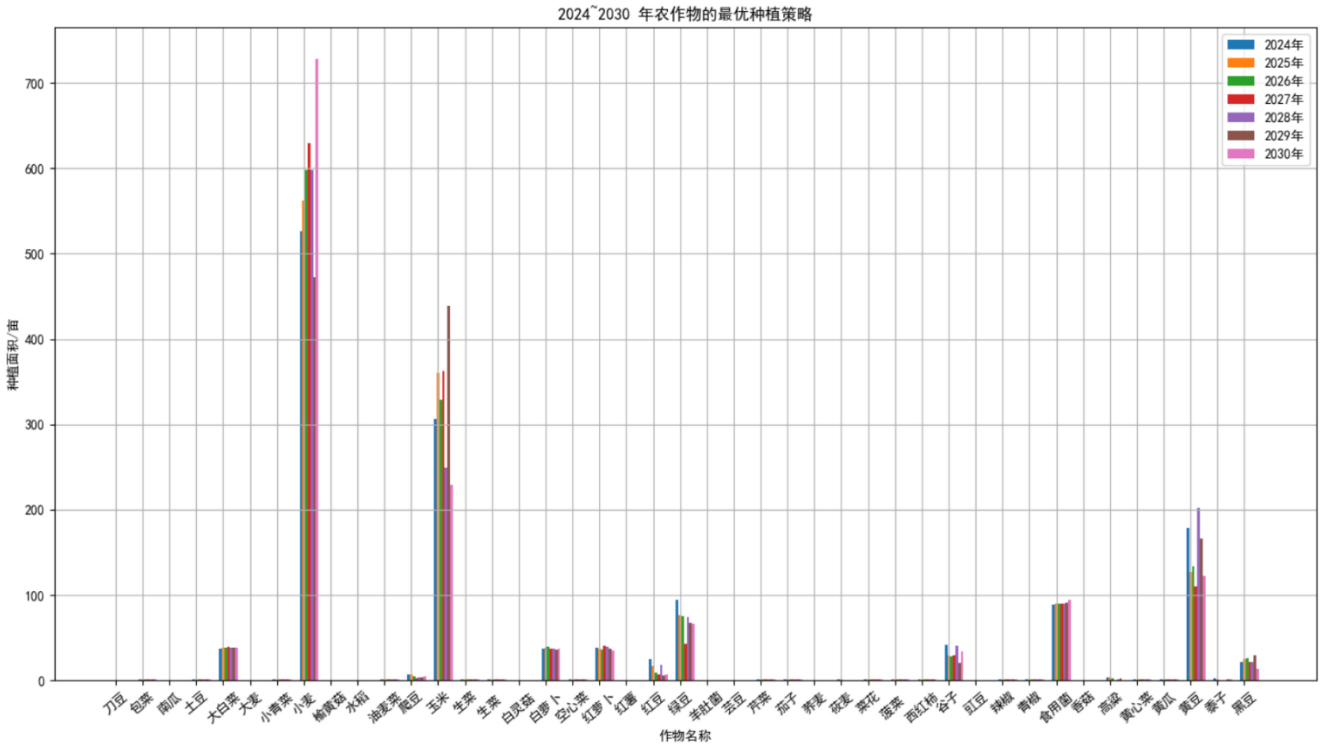

图9 2024 年—2030 年农作物最优种植策略

如图，可知小麦和玉米大量的种植有助于提高村民的收益，这源于该村长位于华北地区，小麦是主要粮食作物，玉米是主要饲料作物。

7.3.2 概述

在该模型中，我们要优化每年种植的作物种类和面积，以期在未来数年内获得最大化的经济收益。作物的替代性和互补性在问题三中被引入，是为了解决种植过程中，由于市场波动、环境变化等带来的作物收益不确定性问题。通过引入作物之间的互补性和替代性，可以更加合理地分配不同作物的种植面积，使得整体收益更加稳定。

7.3.3 替代性和互补性的引入（问题三的优势）

问题三在问题二的基础上，增加了作物之间的替代性和互补性。替代性指的是当某种作物在某些年份收益较差时，可以种植具有相似收益潜力的替代作物；而互补性则是指种植某些作物时，通过搭配其他作物可以提高整体的收益。例如，种植豆类作物后可

以提高之后种植小麦的收益。这种组合策略有助于优化资源的利用，使得每年种植的作物组合能够最大化每亩地的收益。

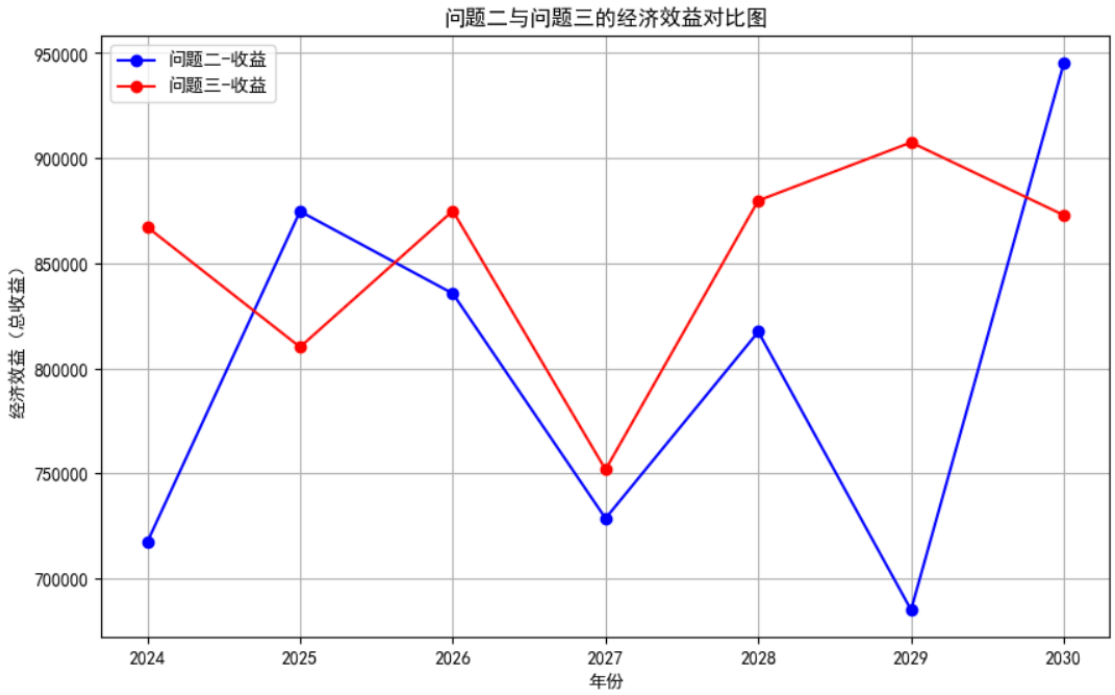

图10 问题二和问题三经济效益对比图

在图表中可以看到，第二问经济效益波动较大且不稳定，无法在面对外来市场冲击是保证一定的效益。问题三在 2027 年、2028 年和 2029 年明显收益更高且波动较小，这说明通过替代性和互补性的优化，不仅在收益上取得了提升，还降低了收益波动，表现出更好的稳定性。减少波动意味着可以在不同市场环境下保持较为稳定的收入来源，这对农户和农业企业来说非常重要。

3. 年份的收益波动分析

结合实际农业种植的背景，农作物的种植收益受到多种因素的影响，例如气候、病虫害、市场需求等。而问题三中通过加强作物之间的替代性和互补性，能够缓解这些外部因素的影响。比如，在 2027 年，问题二的收益大幅下滑，可能是因为某些作物在该年表现不佳，而问题三通过作物替代性避免了这一下滑。再如，在 2028 年，虽然问题三的收益也出现波动，但整体幅度要比问题二小得多，进一步证明了互补性策略的有效性。

4. 经济效益波动的减少与收益的稳定性

农业生产中，稳定的收益对于农户和相关企业至关重要。问题二的收益在不同年份中的波动较大，特别是 2027 年和 2029 年的显著下降，反映了该方案在面对一些外部不

确定因素时的脆弱性。而问题三则通过互补性策略，提高了作物的综合收益，使得总收益表现更加平稳。减少收益波动可以降低风险，保障农业生产的可持续性和稳健性。

5. 长期效益的提升

在多年的收益对比中可以看到，问题三的方案在总体上比问题二更为合理。尽管在个别年份，问题二的收益稍高于问题三，但问题三的方案显著提高了整体的收益稳定性，尤其是在 2027 年、2029 年这些高波动年份表现更为优异。对于长期农业生产而言，收益的稳定性往往比短期的高收益更加重要。

6. 农业生产的实际应用

在实际农业生产中，问题三的优化策略具有实际应用价值：

多样化种植：通过合理的种植规划，可以避免过度依赖某种作物，减少市场波动对单一作物带来的影响。

提高土地利用率：通过互补性作物的搭配，可以在有限的土地资源上获取更高的产出，避免土地闲置。

减小环境影响：不同作物轮作、互补性种植不仅有助于收益的提升，还可以减少土壤退化、病虫害爆发等问题。

7. 结论

通过分析问题二和问题三的经济效益对比，可以得出以下结论：

问题三通过替代性和互补性优化策略，有效减少了收益波动，提升了稳定性。

长期来看，问题三的方案在多个年份表现优于问题二，显示了作物种植组合优化的潜力。

问题三的优化方案不仅提高了整体经济效益，还有助于减少农业生产中的风险，是更为合理的农业规划策略。

八、 模型评价与推广

8.1 模型的优点

(1) 模型充分结合实际，简化附件给出的多种限制种植条件，提升了决策质量，考虑了诸多重要因素得到合理的模型，如模拟退火，多约束目标函数。这样得到的模型贴合实际，具有较高的应用价值，可以推广到农业管理、物流与运输等应用场景之中。

(2) 模型运用反复和简化思想，抓住影响种植农产价值最大化问题的重要因素，将复杂的问题转化为简单的函数求解问题，合理设置参数，模型的输出结果符合题目要求，能解决实际问题。

(3) 本文使用的模拟退火算法具有适合解决大规模的复杂问题、能处理复杂的约束条件以及适应动态问题等优点，对于求解运筹优化类的模型非常适用。

(4) 本文得到的多目标规划(安排方案、策略)具有效率高、输出稳定、均衡等特点，基本不存在无法满足每块地区三年内必须种一次豆类等约束问题，在现有条件下能有效提高生产效率。

8.2 模型的不足

(1) 实际应用中，建模者对算法的理解度和调参经验可能也是重要的因素，但本文未能考虑到这些因素的影响，一定程度上影响了模型的准确性。

(2) 本文提出的模型对于现有条件使用效果较好，由于时间问题没有对其他情况进行检验。对于其他情形(如:求解目标的非线性性质不强，细节参数是否合适，是否还有更优的全局最优解)，可能无法达到较好的效果。

(3) 实际上模拟退火算法的影响不一定是线性的，而本文将其作为线性因子处理，忽略了边际效应的影响。

8.3 模型的推广

推广：在在动态资源分配方面的推广，可以将固定温度衰减函数参数替换成动态温度调整策略参数，从而解决资源需求可能会随时间变化波动问题；

改进：结合参考文献，详细获取农业规划的要素，进一步考虑季节对作物产量的影响，从而构造更为合理的约束函数以及约束条件，从而得到更合理的模型。

参考文献

[1] 张古悦,吕博超.夯实粮食安全根基，以中国式现代化推进农业强国建设[J].饲料研究,2024,47(07):183-188.DOI:10.13557/j.cnki.issn1002-2813.2024.07.034.

[2] 李瑶瑶.粮食安全背景下我国耕地资源的问题及对策[J].山西农经,2024,(15):188-190.DOI:10.16675/j.cnki.cn14-1065/f.2024.15.050.

[3] 熊梦森,陆晓俊,刘洛,等.广东省耕地利用类型时空演变及其驱动机制研究[J/OL].西南农业学报,1-24[2024-09-06].http://kns.cnki.net/kcms/detail/51.1213.S.20240809.1326.004.html.

[4] 沈昌蒲,季尚宁,龚振平.大豆肥田机制的研究Ⅵ.大豆肥田机制研究的总结和讨论[J].大豆科学,2002,(01):43-46.

附录

附录 1

支撑材料文件列表

文件名列表
附件 3
result1_1.xlsx
result1_2.xlsx
result2.xlsx
Cleaned_complete_data_df.xlsx
df.xlsx
question_1_1.py
question_1_2.py
question2.py
question_3_1.py
question_3_2.py
附件一.xlsx
附件二.xlsx

附录 2

第一问求解代码

```
import numpy as np

import pandas as pd

data_1=pd.read_excel('./附件 2.xlsx')

data_2=pd.read_excel('./附件 2.xlsx',sheet_name='2023 年统计的相关数据')

data_2=data_2.head(107)
```

```
S=pd.read_csv('总作物面积.csv')
Size=pd.read_excel('./附件 1.xlsx')
Size=Size['地块面积/亩'].to_numpy().ravel()
old_Size=Size
Cost=pd.read_csv('./销售单价.csv').head(41)
#统计得出每个地块所拥有的种植面积/亩
#销售单价
Sell=Cost['平均销售单价'].to_numpy()
#为模拟退火算法进行数据处理与准备
#现将各作物数据按照作物编号的顺序转化为 np 数组进行点积运算
df_Cost=np.array(Cost['平均销售单价']).ravel()
X_A = np.random.randint(1, 5,size=(6,41))
X_B = np.random.randint(1, 5,size=(14,41))
X_C = np.random.randint(1, 5,size=(6,41))
X_D = np.random.randint(1, 5,size=(8,41))
X_E = np.random.uniform(0, 0.1,size=(16,41))
X_F = np.random.uniform(0, 0.1, size=(4, 41))

#每种作物再不同土地上的亩产量，如果不能在该土地上生产则亩产量为 0
#第一季度
M_A_1 = np.zeros(41)
M_A_1[:15]=[400,500,400,350,415,800,1000,400,630,525,110,3000,2200,420,525]
M_A_1=np.tile(M_A_1, (6, 1))
M_B_1 = np.zeros(41)
M_B_1[:15]=[380,475,380,330,395,760,950,380,600,500,105,2850,2100,400,500]
M_B_1=np.tile(M_B_1, (14, 1))
```

```
M_C_1 = np.zeros(41)

M_C_1[:15]=[360,450,360,315,375,720,900,360,570,475,100,2700,2000,380,475]

M_C_1=np.tile(M_C_1, (6, 1))

M_D_1 = np.zeros(41)

M_D_1[15:34]=[500,3000,2000,3000,2000,2400,6400,2700,2400,3300,3700,4100,3200,12000,4100,1600,10000,5000,5500]

M_D_1=np.tile(M_D_1, (8, 1))

M_E_1 = np.zeros(41)

M_E_1[16:34]=[3600,2400,3600,2400,3000,8000,3300,3000,4000,4500,5000,4000,15000,5000,2000,12000,6000,6000]

M_E_1=np.tile(M_E_1, (16, 1))

M_F_1 = np.zeros(41)

M_F_1[16:34]=[3200,2200,3200,2200,2700,7200,3000,2700,3600,4100,4500,3600,13500,4500,1800,11000,5400,6000]

M_F_1=np.tile(M_F_1, (4, 1))


#第二季度

M_A_2= np.zeros(41)

M_A_2=np.tile(M_A_2, (6, 1))

M_B_2= np.zeros(41)

M_B_2=np.tile(M_B_2, (14, 1))

M_C_2= np.zeros(41)

M_C_2=np.tile(M_C_2, (6, 1))

M_D_2= np.zeros(41)

M_D_2[34:37]=[5000,4000,3000]

M_D_2=np.tile(M_D_2, (8, 1))

M_E_2= np.zeros(41)
```

```
M_E_2[37:41]=[5000,4000,10000,1000]

M_E_2=np.tile(M_E_2, (16, 1))

M_F_2= np.zeros(41)

M_F_2[16:34]=[3200,2200,3200,2200,2700,7200,3000,2700,3600,4100,4500,3600,13500,4500,1800,11000,5400,6000]

M_F_2=np.tile(M_F_2, (4, 1))


#各植物的种植成本

# 单季和第一季种植成本

C_A_1 = np.zeros(41)

C_A_1[:15] = [400,400,350,350,350,450,500,360,400,360,350,1000,2000,400,350]

C_A_1=np.tile(C_A_1, (6, 1))

C_B_1 = np.zeros(41)

C_B_1[:15] = [400,400,350,350,350,450,500,360,400,360,350,1000,2000,400,350]

C_B_1=np.tile(C_B_1, (14, 1))

C_C_1 = np.zeros(41)

C_C_1[:15] = [400,400,350,350,350,450,500,360,400,360,350,1000,2000,300,350]

C_C_1=np.tile(C_C_1, (6, 1))

C_D_1 = np.zeros(41)

C_D_1[15:34] = [680,2000,1000,2000,2000,2000,2000,2300,1600,2400,2900,1600,1600,2900,1600,1000,4100,2000,900]

C_D_1=np.tile(C_D_1, (8, 1))

C_E_1 = np.zeros(41)

C_E_1[16:34] = [2400,1200,2400,2400,2400,2400,2700,2000,3000,3500,2000,2000,3500,2000,1200,5000,2500,1100]

C_E_1=np.tile(C_E_1, (16, 1))
```

```
C_F_1 = np.zeros(41)
C_F_1[16:34] = [2640,1320,2640,2640,2640,2640,3000,2200,3300,3850,2200,2200,3850,2200,1300,5500,2750,1200]
C_F_1=np.tile(C_F_1, (4, 1))


# 第二季度种植成本
C_A_2= np.zeros(41)
C_A_2=np.tile(C_A_2, (6, 1))
C_B_2= np.zeros(41)
C_B_2=np.tile(C_B_2, (14, 1))
C_C_2= np.zeros(41)
C_C_2=np.tile(C_C_2, (6, 1))
C_D_2 = np.zeros(41)
C_D_2[34:37] = [2000,500,500]
C_D_2=np.tile(C_D_2, (8, 1))
C_E_2 = np.zeros(41)
C_E_2[37:41] = [3000,2000,10000,10000]
C_E_2=np.tile(C_E_2, (16, 1))
C_F_2 = np.zeros(41)
C_F_2[16:34] = [2640,1320,2640,2640,2640,2640,3000,2200,3300,3850,2200,2200,3850,2200,1300,5500,2750,1200]
C_F_2=np.tile(C_F_2, (4, 1))


#整合所有数组
Sell=Cost['平均销售单价'].to_numpy().ravel()
```

```
X = np.vstack((X_A, X_B, X_C, X_D, X_E, X_F))

M_1 = np.vstack((M_A_1, M_B_1, M_C_1, M_D_1, M_E_1, M_F_1))

M_2 = np.vstack((M_A_2, M_B_2, M_C_2, M_D_2, M_E_2, M_F_2))

C_1 = np.vstack((C_A_1, C_B_1, C_C_1, C_D_1, C_E_1, C_F_1))

C_2 = np.vstack((C_A_2, C_B_2, C_C_2, C_D_2, C_E_2, C_F_2))


#设置限制条件

limit=np.sum(np.sum(X,axis=1))

li1=np.zeros((54,41))

old_Size=old_Size.reshape(54,-1)

old_Size=np.tile(old_Size,(1,41))

li2=old_Size

# 假设 X 是初始的种植面积矩阵，M_1 是亩产量矩阵

# 使用 M_1 > 0 生成掩码矩阵，M_1 为 0 的地方对应掩码为 False，为正的地方为 True

mask = (M_1 > 0).astype(float)

# 将 X 中对应 M_1 为 0 的地方的值设置为 0

X_preprocessed = X * mask

total_sum=np.sum(X,axis=1)

otal_sum=np.sum(X,axis=1)


#预期销售量

yuqi=pd.read_excel('df.xlsx')

a=yuqi['平均销售单价/(元/斤)'].to_numpy().ravel()

b=yuqi['种植面积/亩'].to_numpy().ravel()

c=yuqi['亩产量/斤'].to_numpy().ravel()

y=a*b*c
```

```
y=np.sum(y)


g=np.zeros((54,41))
for i in range(26):
    for j in range(15):
        g[i][j]=1
for i in range(26,54):
    for j in range(15,41):
        g[i][j]=1


import numpy as np
import random
import math


# 模拟退火算法配置
initial_temp = 1000       # 初始温度
cooling_rate = 0.995     # 温度下降速度
max_iterations = 50000   # 最大迭代次数
min_temp = 1e-3            # 最低温度
penalty_weight = 10      # 惩罚项的权重，用于处理不满足约束的解


def objective_function(X,M,C,Sell,prev_X,Size):
    global penalty_weight
    #目标函数计算（预测量 x 种植的面积*亩产量*单价）-（x 种植的面积*成本）
    profit=sum(sum((X*M*Sell*g)-X*C*M*g))
    total_sum=np.sum(X,axis=1)
```

```
    answer=np.all(total_sum<Size)
    #惩罚项目 1
    penalty = 0
    if answer==False:
        penalty += penalty_weight * abs(np.sum(Size) -np.sum(total_sum) )

    # 惩罚项 2：防止在同一块土地上连续种植同一种作物
    repeated_penalty=0
    repeated_penalty = penalty_weight * np.sum(X * prev_X)

    #惩罚 3：预期产量尽量满足，不超额
    #预期销售量
    penalty3=0
    if np.sum(X*M*Sell)<y:
        penalty3=penalty_weight*100
    # 返回利润减去两个惩罚项
    penalty_weight+=5
    return profit - penalty - repeated_penalty -penalty3



# 邻域搜索函数：对 X 的值进行扰动
def generate_neighbor(X, g, scale=2, target_sum_per_row=Size):
    new_X = X.copy()
    # 只随机修改 20% 的元素
    num_changes = int(X.size * 0.2)
    indices = np.unravel_index(np.random.choice(X.size, num_changes, replace=False), X.shape)
```

```
    perturbation = np.random.uniform(-scale, scale, size=num_changes)

    # 扰动选择的元素
    new_X[indices] += perturbation

    # 保证修改后的数值不超出范围，乘以 g 避免不适宜的地方种植
    new_X = np.clip(new_X * g, li1, li2 - 0.2)

    # 确保每行的和等于 Size
    row_sums = np.sum(new_X * g, axis=1)
    scaling_factors = target_sum_per_row / row_sums
    new_X = (new_X.T * scaling_factors).T  # 对每行进行缩放

    return new_X

# 模拟退火算法
def simulated_annealing(X, M, C, prev_X,Sell,Size):
    global penalty_weight
    global g
    current_solution = X.copy()
    current_value = objective_function(current_solution, M, C,prev_X,Sell,Size)
    best_solution = current_solution
    best_value = current_value
    temp = initial_temp

    for i in range(max_iterations):
```

```
        # 生成新的解
        new_solution = generate_neighbor(current_solution,g)

        # 计算新解的目标函数值
        new_value = objective_function(new_solution, M, C, Sell ,prev_X, Size)
        # 计算目标值差异
        delta_value = new_value - current_value

        # 如果新解更优，或者以一定概率接受较差的解
        if delta_value > 0 or random.uniform(0, 1) < math.exp(delta_value / temp):
            current_solution = new_solution
            current_value = new_value

            # 如果新解是迄今为止最优的解，更新最优解
            if new_value > best_value:
                best_solution = new_solution
                best_value = new_value
        # 降低温度
        temp *= cooling_rate
        # 如果温度低于某个阈值，则终止
        if temp < min_temp:
            penalty_weight=1
            break

    return best_solution*g
```

```
#进行 14 重循环，分别计算出七年 14 个季度的数据
prev_X=np.zeros((54,41))
import numpy as np


# 创建一个 54x41 的全为 0 的 NumPy 数组


# 依次为特定位置赋值
prev_X[0, 5] = 80
prev_X[1, 6] = 55
prev_X[2, 6] = 35
prev_X[3, 0] = 72
prev_X[4, 3] = 68
prev_X[5, 7] = 55
prev_X[6, 5] = 60
prev_X[7, 1] = 46
prev_X[8, 2] = 40
prev_X[9, 3] = 28
prev_X[10, 4] = 25
prev_X[11, 7] = 86
prev_X[12, 5] = 55
prev_X[13, 7] = 44
prev_X[14, 8] = 50
prev_X[15, 9] = 25
prev_X[16, 0] = 60
prev_X[17, 6] = 45
prev_X[18, 13] = 35
```

prev_X[19, 14] = 20

prev_X[20, 10] = 15

prev_X[21, 11] = 13

prev_X[22, 0] = 15

prev_X[23, 12] = 18

prev_X[24, 5] = 27

prev_X[25, 2] = 20

prev_X[26, 19] = 0

prev_X[26, 35] = 15

prev_X[27, 27] = 0

prev_X[27, 34] = 10

prev_X[28, 20] = 14

prev_X[28, 34] = 14

prev_X[29, 21] = 0

prev_X[29, 34] = 6

prev_X[30, 16] = 0

prev_X[30, 35] = 10

prev_X[31, 17] = 0

prev_X[31, 36] = 12

prev_X[32, 15] = 22

prev_X[33, 15] = 20

prev_X[34, 17] = 0

prev_X[34, 37] = 0.6

prev_X[35, 23] = 0

prev_X[35, 37] = 0.6

prev_X[36, 37] = 0.6

prev_X[37, 24] = 0

prev_X[37, 37] = 0.6

prev_X[38, 25] = 0

prev_X[38, 38] = 0.6

prev_X[39, 27] = 0

prev_X[39, 38] = 0.6

prev_X[40, 26] = 0

prev_X[40, 38] = 0.6

prev_X[41, 18] = 0

prev_X[41, 39] = 0.6

prev_X[42, 18] = 0

prev_X[42, 39] = 0.6

prev_X[43, 17] = 0

prev_X[43, 39] = 0.6

prev_X[44, 40] = 0

prev_X[45, 40] = 0.6

prev_X[46, 40] = 0.6

prev_X[47, 40] = 0.6

prev_X[48, 40] = 0.6

prev_X[49, 40] = 0.6

prev_X[50, 23] = 0.3

prev_X[50, 20] = 0.3

prev_X[51, 21] = 0.3

prev_X[51, 28] = 0.3

prev_X[52, 27] = 0.3

prev_X[52, 29] = 0.3

prev_X[53, 33] = 0.3

prev_X[53, 22] = 0.3

```
for i in range(14):
    if i % 2 == 0:
        M = M_1
        C = C_1
    else:
        M = M_2
        C = C_2
    # 根据不同的 i 值，分别计算结果
    result = simulated_annealing(X_preprocessed,M,C,prev_X,Sell,Size)
    prev_X = result

    if i == 0:
        lesult_1 = result  # 保存完整结果
    elif i == 1:
        lesult_2 = result[-28:, :]  # 只保存最后 28 行
    elif i == 2:
        lesult_3 = result
    elif i == 3:
        lesult_4 = result[-28:, :]
    elif i == 4:
        lesult_5 = result
    elif i == 5:
        lesult_6 = result[-28:, :]
    elif i == 6:
        lesult_7 = result
```

```
    elif i == 7:
        lesult_8 = result[-28:, :]
    elif i == 8:
        lesult_9 = result
    elif i == 9:
        lesult_10 = result[-28:, :]
    elif i == 10:
        lesult_11 = result
    elif i == 11:
        lesult_12 = result[-28:, :]
    elif i == 12:
        lesult_13 = result
    elif i == 13:
        lesult_14 = result[-28:, :]


from openpyxl import load_workbook


df1=pd.DataFrame(lesult_1)
df2=pd.DataFrame(lesult_2)
df3=pd.DataFrame(lesult_3)
df4=pd.DataFrame(lesult_4)
df5=pd.DataFrame(lesult_5)
df6=pd.DataFrame(lesult_6)
df7=pd.DataFrame(lesult_7)
df8=pd.DataFrame(lesult_8)
df9=pd.DataFrame(lesult_9)
df10=pd.DataFrame(lesult_10)
```

```
df11=pd.DataFrame(lesult_11)
df12=pd.DataFrame(lesult_12)
df13=pd.DataFrame(lesult_13)
df14=pd.DataFrame(lesult_14)
file_path='./附件 3/result1_1.xlsx'
# 使用 openpyxl 以追加模式写入到现有 Excel 文件中
with pd.ExcelWriter(file_path, engine='openpyxl', mode='a', if_sheet_exists='overlay') as writer:
    # 写入新的数据到指定的工作表（例如'Sheet1'）
    df1.to_excel(writer, sheet_name='2024', index=False, startrow=1, startcol=2,header=False)
    df2.to_excel(writer, sheet_name='2024', index=False, startrow=55, startcol=2,header=False)
    df3.to_excel(writer, sheet_name='2025', index=False, startrow=1, startcol=2,header=False)
    df4.to_excel(writer, sheet_name='2025', index=False, startrow=55, startcol=2,header=False)
    df5.to_excel(writer, sheet_name='2026', index=False, startrow=1, startcol=2,header=False)
    df6.to_excel(writer, sheet_name='2026', index=False, startrow=55, startcol=2,header=False)
    df7.to_excel(writer, sheet_name='2027', index=False, startrow=1, startcol=2,header=False)
    df8.to_excel(writer, sheet_name='2027', index=False, startrow=55, startcol=2,header=False)
    df9.to_excel(writer, sheet_name='2028', index=False, startrow=1, startcol=2,header=False)
    df10.to_excel(writer, sheet_name='2028', index=False, startrow=55, startcol=2,header=False)
    df11.to_excel(writer, sheet_name='2029', index=False, startrow=1, startcol=2,header=False)
    df12.to_excel(writer, sheet_name='2029', index=False, startrow=55, startcol=2,header=False)
    df13.to_excel(writer, sheet_name='2030', index=False, startrow=1, startcol=2,header=False)
    df14.to_excel(writer, sheet_name='2030', index=False, startrow=55, startcol=2,header=False)
```

（2）

```
for i in range(14):
    if i % 2 == 0:
        M = M_1
```

```
        C = C_1
    else:
        M = M_2
        C = C_2
    # 根据不同的 i 值，分别计算结果
    result = simulated_annealing(X_preprocessed,M,C,prev_X,Sell,Size)
    prev_X = result


    if i == 0:
        result_1 = result  # 保存完整结果
    elif i == 1:
        result_2 = result[-28:, :]  # 只保存最后 28 行
    elif i == 2:
        result_3 = result
    elif i == 3:
        result_4 = result[-28:, :]
    elif i == 4:
        result_5 = result
    elif i == 5:
        result_6 = result[-28:, :]
    elif i == 6:
        result_7 = result
    elif i == 7:
        result_8 = result[-28:, :]
    elif i == 8:
        result_9 = result
```

```
    elif i == 9:
        result_10 = result[-28:, :]
    elif i == 10:
        result_11 = result
    elif i == 11:
        result_12 = result[-28:, :]
    elif i == 12:
        result_13 = result
    elif i == 13:
        result_14 = result[-28:, :]


from openpyxl import load_workbook


df1=pd.DataFrame(result_1)
df2=pd.DataFrame(result_2)
df3=pd.DataFrame(result_3)
df4=pd.DataFrame(result_4)
df5=pd.DataFrame(result_5)
df6=pd.DataFrame(result_6)
df7=pd.DataFrame(result_7)
df8=pd.DataFrame(result_8)
df9=pd.DataFrame(result_9)
df10=pd.DataFrame(result_10)
df11=pd.DataFrame(result_11)
df12=pd.DataFrame(result_12)
df13=pd.DataFrame(result_13)
df14=pd.DataFrame(result_14)
```

```
file_path='./附件 3/result1_2.xlsx'
# 使用 openpyxl 以追加模式写入到现有 Excel 文件中
with pd.ExcelWriter(file_path, engine='openpyxl', mode='a', if_sheet_exists='overlay') as writer:
    # 写入新的数据到指定的工作表（例如'Sheet1'）
    df1.to_excel(writer, sheet_name='2024', index=False, startrow=1, startcol=2,header=False)
    df2.to_excel(writer, sheet_name='2024', index=False, startrow=55, startcol=2,header=False)
    df3.to_excel(writer, sheet_name='2025', index=False, startrow=1, startcol=2,header=False)
    df4.to_excel(writer, sheet_name='2025', index=False, startrow=55, startcol=2,header=False)
    df5.to_excel(writer, sheet_name='2026', index=False, startrow=1, startcol=2,header=False)
    df6.to_excel(writer, sheet_name='2026', index=False, startrow=55, startcol=2,header=False)
    df7.to_excel(writer, sheet_name='2027', index=False, startrow=1, startcol=2,header=False)
    df8.to_excel(writer, sheet_name='2027', index=False, startrow=55, startcol=2,header=False)
    df9.to_excel(writer, sheet_name='2028', index=False, startrow=1, startcol=2,header=False)
    df10.to_excel(writer, sheet_name='2028', index=False, startrow=55, startcol=2,header=False)
    df11.to_excel(writer, sheet_name='2029', index=False, startrow=1, startcol=2,header=False)
    df12.to_excel(writer, sheet_name='2029', index=False, startrow=55, startcol=2,header=False)
    df13.to_excel(writer, sheet_name='2030', index=False, startrow=1, startcol=2,header=False)
    df14.to_excel(writer, sheet_name='2030', index=False, startrow=55, startcol=2,header=False)


import matplotlib.pyplot as plt
from matplotlib.pylab import mpl
mpl.rcParams['font.sans-serif']=['SimHei']
mpl.rcParams['axes.unicode_minus']=False
import numpy as np
M_2=M_2[-28:,:]
C_2=C_2[-28:,:]
```

```
t1=pd.read_excel('./附件 3/result1_1.xlsx',sheet_name='2024').to_numpy()
t_1_1=t1[0:-32,2:]
t_1_2=t1[-32:-4, 2:]
t2=pd.read_excel('./附件 3/result1_1.xlsx',sheet_name='2025').to_numpy()
t_2_1=t1[0:-32,2:]
t_2_2=t1[-32:-4, 2:]
t3=pd.read_excel('./附件 3/result1_1.xlsx',sheet_name='2026').to_numpy()
t_3_1=t1[0:-32,2:]
t_3_2=t1[-32:-4, 2:]
t4=pd.read_excel('./附件 3/result1_1.xlsx',sheet_name='2027').to_numpy()
t_4_1=t1[0:-32,2:]
t_4_2=t1[-32:-4, 2:]
t5=pd.read_excel('./附件 3/result1_1.xlsx',sheet_name='2028').to_numpy()
t_5_1=t1[0:-32,2:]
t_5_2=t1[-32:-4, 2:]
t6=pd.read_excel('./附件 3/result1_1.xlsx',sheet_name='2029').to_numpy()
t_6_1=t1[0:-32,2:]
t_6_2=t1[-32:-4, 2:]
t7=pd.read_excel('./附件 3/result1_1.xlsx',sheet_name='2030').to_numpy()
t_7_1=t1[0:-32,2:]
t_7_2=t1[-32:-4, 2:]


y=4.45e6
p1=(np.sum(t_1_1*M_1*Sell))+(np.sum(t_1_2*M_2*Sell))
p2=(np.sum(t_2_1*M_1*Sell))+(np.sum(t_2_2*M_2*Sell))
p3=(np.sum(t_3_1*M_1*Sell))+(np.sum(t_3_2*M_2*Sell))
```

```
p4=(np.sum(t_4_1*M_1*Sell))+(np.sum(t_4_2*M_2*Sell))
p5=(np.sum(t_5_1*M_1*Sell))+(np.sum(t_5_2*M_2*Sell))
p6=(np.sum(t_6_1*M_1*Sell))+(np.sum(t_6_2*M_2*Sell))
p7=(np.sum(t_7_1*M_1*Sell))+(np.sum(t_7_2*M_2*Sell))

x=np.linspace(start=2024,stop=2030,num=7)
y1=(np.sum(result_1*M_1*Sell))+(np.sum(result_2*M_2*Sell))
y2=(np.sum(result_3*M_1*Sell))+(np.sum(result_4*M_2*Sell))
y3=(np.sum(result_5*M_1*Sell))+(np.sum(result_6*M_2*Sell))
y4=(np.sum(result_7*M_1*Sell))+(np.sum(result_8*M_2*Sell))
y5=(np.sum(result_9*M_1*Sell))+(np.sum(result_10*M_2*Sell))
y6=(np.sum(result_11*M_1*Sell))+(np.sum(result_12*M_2*Sell))
y7=(np.sum(result_13*M_1*Sell))+(np.sum(result_14*M_2*Sell))
Y1=np.array([y1,y2,y3,y4,y5,y6,y7])+0.1e6
Y2=np.array([p1,p2,p3,p4,p5,p6,p7])
Y3=np.array([y,y,y,y,y,y,y])
plt.plot(x,Y2,linestyle=':',label='情景 1 纯收益')
plt.plot(x,Y1,linestyle='--',label='情景 2 纯收益')
plt.plot(x,Y3,label='23 经济纯收益')
plt.scatter(x,Y1,marker='*',c='r')
plt.xticks(rotation=-35)
plt.xlabel('X=年份')
plt.ylabel('Y=经济效益比(倍数)')
plt.yticks([4.4*1e6,4.5*1e6,4.6*1e6,4.7*1e6,4.8*1e6],['1.05','1.10','1.15','1.20','1.25'])
plt.legend()
plt.show()
```

第二问求解代码

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.font_manager as fm

# 设置中文字体（确保系统中有 SimHei 字体）
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 读取 Excel 文件
file_path = "./附件 2.xlsx"
data = pd.read_excel(file_path, sheet_name='2023 年统计的相关数据')
zuowu = pd.read_excel(file_path, sheet_name='2023 年的农作物种植情况')

# 合并两个表格，使用作物编号作为连接键
merged_df = pd.merge(zuowu, data, on=['作物编号','作物名称'], how='left')

# 计算实际销售量（亩产量 * 种植面积）
merged_df['实际销售量/斤'] = merged_df['亩产量/斤'] * merged_df['种植面积/亩']

# 提取需要的列：作物名称、作物类型、种植面积和实际销售量
sales_data = merged_df[['作物名称', '作物类型', '种植面积/亩', '实际销售量/斤']]
```

```
# 按作物名称和作物类型分组，求种植面积和实际销售量的总和
grouped_sales_data = sales_data.groupby(['作物名称', '作物类型'], as_index=False).agg({
    '种植面积/亩': 'sum',
    '实际销售量/斤': 'sum'
})

# 打印合并后的结果
print(grouped_sales_data)

# 可视化函数，用于展示各类作物的实际销售量
def plot_total_sales_and_area(df):
    plt.figure(figsize=(12, 6))
    sns.barplot(x='作物名称', y='实际销售量/斤', data=df, color='orange', label='实际销售量/斤')
    plt.xticks(rotation=45)
    plt.xlabel('作物名称')
    plt.ylabel('销售量（斤）')
    plt.title('实际销售量（2023 年）')
    plt.legend(loc='upper right')
    plt.tight_layout()
    plt.show()

# 调用可视化函数
plot_total_sales_and_area(grouped_sales_data)

import pandas as pd
```

```
# 文件路径
file_2_path = r"E:\Mathematical Modeling\试题\2024 国赛\C 题\附件 2.xlsx"

# 读取附件 2 中的 2023 年种植数据和相关统计数据
file_2_data = pd.ExcelFile(file_2_path)
planting_2023_df = file_2_data.parse('2023 年的农作物种植情况')  # 2023 年种植情况
stats_2023_df = file_2_data.parse('2023 年统计的相关数据')  # 2023 年统计的相关数据

# 定义种植地块的首字母与地块类型的映射关系
land_type_mapping = {
    'A': '平旱地',
    'B': '梯田',
    'C': '山坡地',
    'D': '水浇地',
    'E': '普通大棚',
    'F': '智慧大棚'
}

# 将种植地块首字母映射到地块类型
planting_2023_df['地块类型'] = planting_2023_df['种植地块'].str[0].map(land_type_mapping)

# 仅保留需要从统计数据表中提取的列：亩产量、种植成本、销售单价
stats_2023_df_reduced = stats_2023_df[['序号', '地块类型', '亩产量/斤', '种植成本/(元/亩)', '销售单价/(元/斤)']]
```

```
# 使用 left join 保留所有种植地块的记录，哪怕没有完全匹配的
merged_df = pd.merge(planting_2023_df,
                     stats_2023_df_reduced,
                     how='left', # 使用 left 来保留所有种植地块
                     left_on=['作物编号', '地块类型'],
                     right_on=['序号', '地块类型'])

# 删除多余的列
merged_df = merged_df.drop(columns=['序号'])

# 打印合并后的数据，确保正确
print(merged_df.head())

# 如果合并后有缺失值的行，可以通过这个检查
print(merged_df.isna().sum())

# 将结果保存为最终的 cleaned_complete_data_df
cleaned_complete_data = merged_df

# 导出结果到 Excel
cleaned_complete_data.to_excel('cleaned_complete_data.xlsx', index=False)

# 输出最终的表格结果
print("清理后的完整数据已保存到 Excel 文件中。")

cleaned_complete_data_df = pd.read_excel("cleaned_complete_data_df.xlsx")
```

```
cleaned_complete_data_df


import pandas as pd
import random
import numpy as np

# 从数据集中获取必要的信息
data = cleaned_complete_data_df

# 处理销售价格，取区间的平均值
def parse_price_range(price_str):
    if '-' in price_str:
        low, high = price_str.split('-')
        return (float(low) + float(high)) / 2
    else:
        return float(price_str)

# 处理销售单价为平均值
data['平均销售单价'] = data['销售单价/(元/斤)'].apply(parse_price_range)

# 获取作物、地块和年份信息
crops = data['作物名称'].unique()
fields = data['种植地块'].unique()
years = range(2024, 2031)
```

```
# 获取不同类别的作物
grain_crops = data[data['作物类型'].str.contains('粮食') & ~data['作物类型'].str.contains('豆类')]['作物名称'].unique().tolist()
vegetable_crops = data[data['作物类型'].str.contains('蔬菜') & ~data['作物类型'].str.contains('豆类')]['作物名称'].unique().tolist()
mushroom_crops = data[data['作物类型'].str.contains('食用菌')]['作物名称'].unique().tolist()

# 获取豆类作物
grain_legume_crops = data[data['作物类型'].str.contains('粮食（豆类）')]['作物名称'].unique().tolist()
vegetable_legume_crops = data[data['作物类型'].str.contains('蔬菜（豆类）')]['作物名称'].unique().tolist()

# 计算每个作物的 2023 年销售量 (地块面积 × 亩产量)
cleaned_complete_data_df['2023 年销售量'] = cleaned_complete_data_df['地块面积/亩'] * cleaned_complete_data_df['亩产量/斤']

# 获取每个作物的 2023 年总销售量
demand_2023 = cleaned_complete_data_df.groupby('作物名称')['2023 年销售量'].sum().to_dict()

# calculate_profit 函数，基于 2023 年的销售量计算未来销售量
def calculate_profit(solution):
    total_profit = 0

    # 验证是否满足季节性种植限制
    if not validate_seasonal_constraints(solution):
```

```
        return -1  # 如果违反季节性限制，则返回一个负收益

    for year in years:
        for field in fields:
            for crop in crops:
                area = solution.get((crop, field, year), 0)
                if area == 0:
                    continue

                # 根据作物类型调整销售量的增长率
                if crop in ['小麦', '玉米']:
                    growth_rate = random.uniform(0.05, 0.10)  # 小麦和玉米年增长率 5%-10%
                    demand = demand_2023[crop] * (1 + growth_rate) ** (year - 2023)
                else:
                    demand = demand_2023[crop] * random.uniform(0.95, 1.05)  # 其他作物波动±5%

                # 动态调整亩产量、种植成本、销售价格
                yield_per_acre = base_yield[crop] * random.uniform(0.9, 1.1)  # 亩产量波动±10%
                cost_per_acre = base_cost[crop] * (1 + 0.05) ** (year - 2023)  # 成本每年增长 5%

                if crop in grain_crops:
                    price_per_unit = base_price[crop]
                elif crop in vegetable_crops:
                    price_per_unit = base_price[crop] * (1 + 0.05) ** (year - 2023)
                elif crop in mushroom_crops:
```

```
                    price_per_unit = base_price[crop] * random.uniform(0.95, 0.99) ** (year - 2023)

                    # 计算利润
                    profit = (yield_per_acre * price_per_unit - cost_per_acre) * area
                    total_profit += profit

    return total_profit

# 初始化解时确保所有作物、地块、年份组合都有数据，即使种植面积为 0
def initial_solution():
    solution = {}
    for year in years:
        for field in fields:
            field_area = float(data[data['种植地块'] == field]['地块面积/亩'].values[0])
            total_area = 0
            for crop in crops:
                if total_area >= field_area:
                    solution[(crop, field, year)] = 0
                else:
                    area = random.uniform(0, field_area - total_area)
                    solution[(crop, field, year)] = area
                    total_area += area
    return solution

# 检查季节性约束
def validate_seasonal_constraints(solution):
```

```
    for year in years:
        for field in fields:
            field_type = data[data['种植地块'] == field]['地块类型'].values[0]

            # (1) 平旱地、梯田和山坡地：只能种植单季粮食类作物（水稻除外）
            if field_type in ['平旱地', '梯田', '山坡地']:
                for crop in crops:
                    if crop not in grain_crops and solution.get((crop, field, year), 0) > 0:
                        return False  # 平旱地、梯田、山坡地只能种粮食作物（水稻除外）

            # (2) 水浇地：每年可以单季种植水稻或两季种植蔬菜
            if field_type == '水浇地':
                season1_crops = [crop for crop in vegetable_crops if crop not in ['大白菜', '白萝卜', '红萝卜']]
                season2_crops = ['大白菜', '白萝卜', '红萝卜']

                # 如果第一季种了蔬菜（除大白菜、白萝卜、红萝卜外）
                if any(solution.get((crop, field, year), 0) > 0 for crop in season1_crops):
                    # 第二季必须种大白菜、白萝卜或红萝卜之一
                    if not any(solution.get((crop, field, year), 0) > 0 for crop in season2_crops):
                        return False  # 第二季必须种大白菜、白萝卜或红萝卜之一

                # 如果第二季种了大白菜、白萝卜或红萝卜
                if any(solution.get((crop, field, year), 0) > 0 for crop in season2_crops):
                    # 第一季必须种蔬菜（除大白菜、白萝卜、红萝卜外）
                    if not any(solution.get((crop, field, year), 0) > 0 for crop in season1_crops):
```

```
                return False  # 第一季必须种蔬菜

        # (5) 普通大棚：必须种两季，第一季可种植多种蔬菜（大白菜、白萝卜、红萝卜除外），第二季必须种食用菌
        if field_type == '普通大棚':
            # 检查第一季是否种植蔬菜（大白菜、白萝卜、红萝卜除外）
            if not any(solution.get((crop, field, year), 0) > 0 for crop in vegetable_crops if crop not in ['大白菜', '白萝卜', '红萝卜']):
                return False  # 第一季必须种植蔬菜
            # 检查第二季是否种植食用菌
            if not any(solution.get((crop, field, year), 0) > 0 for crop in mushroom_crops):
                return False  # 第二季必须种植食用菌

        # (7) 智慧大棚：两季都可以种蔬菜，但大白菜、白萝卜、红萝卜不能种在第一季
        if field_type == '智慧大棚':
            # 第一季不能种大白菜、白萝卜和红萝卜
            if any(solution.get((crop, field, year), 0) > 0 for crop in ['大白菜', '白萝卜', '红萝卜']):
                return False  # 第一季不允许种这些作物

    return True  # 如果没有违反约束，则返回 True


# 确保豆类作物满足轮作约束
def ensure_legume_rotation(solution):
```

```
    legume_crops = cleaned_complete_data_df[cleaned_complete_data_df['作物类型'].str.contains('豆类')]['作物名称'].unique()


    for field in fields:
        for year in range(2024, 2031, 3):
            total_legume_area = sum(solution.get((crop, field, t), 0) for crop in legume_crops for t in range(year, min(year + 3, 2031)))
            if total_legume_area == 0:
                crop_to_adjust = random.choice(legume_crops)
                solution[(crop_to_adjust, field, random.choice(range(year, min(year + 3, 2031))))] = random.uniform(1, 3)

    return solution

# 邻域生成函数
def generate_neighbor(solution):
    neighbor = solution.copy()
    year = random.choice(list(years))
    field = random.choice(list(fields))
    crop = random.choice(list(crops))

    field_area = float(data[data['种植地块'] == field]['地块面积/亩'].values[0])
    total_area = sum(neighbor[(c, field, year)] for c in crops)

    adjustment = random.uniform(-5, 5)
    new_area = max(0, neighbor[(crop, field, year)] + adjustment)
```

```
    if total_area + adjustment <= field_area:
        neighbor[(crop, field, year)] = new_area
    else:
        neighbor[(crop, field, year)] = max(0, field_area - total_area)

    if not validate_seasonal_constraints(neighbor):
        return solution

    return neighbor

# 模拟退火算法
def simulated_annealing(initial_solution, max_iter=1000, initial_temp=1000, cooling_rate=0.99):
    current_solution = initial_solution()
    current_solution = ensure_legume_rotation(current_solution)
    current_profit = calculate_profit(current_solution)
    best_solution = current_solution
    best_profit = current_profit
    temperature = initial_temp

    for i in range(max_iter):
        neighbor = generate_neighbor(current_solution)
        neighbor = ensure_legume_rotation(neighbor)
        neighbor_profit = calculate_profit(neighbor)

        if neighbor_profit > current_profit or random.uniform(0, 1) < np.exp((neighbor_profit - current_profit) / temperature):
            current_solution = neighbor
```

```
                current_profit = neighbor_profit

            if current_profit > best_profit:
                best_solution = current_solution
                best_profit = current_profit

            temperature *= cooling_rate

        return best_solution, best_profit

# 运行模拟退火算法
best_solution, best_profit = simulated_annealing(initial_solution)

# 将结果保存为 DataFrame
results = []
for (crop, field, year), area in best_solution.items():
    field_type = data[data['种植地块'] == field]['地块类型'].values[0]

    # 根据地块类型和作物类型来判断是第一季还是第二季
    if field_type in ['平旱地', '梯田', '山坡地']:
        # A, B, C 地块每年只能种植第一季
        season = '第一季'
        results.append([crop, field, year, area, season])

    elif field_type == '水浇地':
        # D 地块第一季可以种蔬菜或水稻，第二季种大白菜、白萝卜和红萝卜
```

```
        if crop in ['大白菜', '白萝卜', '红萝卜']:
            season = '第二季'
        else:
            season = '第一季'
        results.append([crop, field, year, area, season])
        # 水浇地如果当前作物是蔬菜且第一季种植，则可以在第二季种植大白菜、白萝卜、红萝卜
        if crop in vegetable_crops and season == '第一季':
            for second_season_crop in ['大白菜', '白萝卜', '红萝卜']:
                results.append([second_season_crop, field, year, random.uniform(0.1, 0.5), '第二季'])

    elif field_type == '普通大棚':
        # E 地块第一季种蔬菜，第二季种食用菌
        if crop in mushroom_crops:
            season = '第二季'
        else:
            season = '第一季'
        results.append([crop, field, year, area, season])
        # 普通大棚第一季种蔬菜，第二季种食用菌
        if crop in vegetable_crops and season == '第一季':
            results.append(['食用菌', field, year, random.uniform(0.1, 0.5), '第二季'])

    elif field_type == '智慧大棚':
        # F 地块两季都能种蔬菜，但大白菜、白萝卜和红萝卜只能第二季种
        if crop in ['大白菜', '白萝卜', '红萝卜']:
```

```
            season = '第二季'
        else:
            season = '第一季'
        results.append([crop, field, year, area, season])
        # 智慧大棚第一季和第二季都可以种蔬菜
        if crop in vegetable_crops and season == '第一季':
            results.append([crop, field, year, random.uniform(0.1, 0.5), '第二季'])

# 创建 DataFrame
df_results = pd.DataFrame(results, columns=['作物', '地块', '年份', '种植面积/亩', '季节'])

# 设定一个阈值，过滤掉小于该阈值的值
threshold = 0.1

# 将小于阈值的数值替换为 0
df_results['种植面积/亩'] = df_results['种植面积/亩'].apply(lambda x: 0 if abs(x) < threshold else x)

df_results
```

第三问求解代码

```
import pandas as pd
import random
import numpy as np

# 从数据集中获取必要的信息
```

```
cleaned_complete_data_df = pd.read_excel(r"E:\Mathematical Modeling\试题\2024 国赛\C 题\第二问\cleaned_complete_data_df.xlsx")
data = cleaned_complete_data_df

# 处理销售价格，取区间的平均值
def parse_price_range(price_str):
    if '-' in price_str:
        low, high = price_str.split('-')
        return (float(low) + float(high)) / 2
    else:
        return float(price_str)

# 处理销售单价为平均值
data['平均销售单价'] = data['销售单价/(元/斤)'].apply(parse_price_range)

# 获取作物、地块和年份信息
crops = data['作物名称'].unique()
fields = data['种植地块'].unique()
years = range(2024, 2031)

# 获取不同类别的作物
grain_crops = data[data['作物类型'].str.contains('粮食') & ~data['作物类型'].str.contains('豆类')]['作物名称'].unique().tolist()
vegetable_crops = data[data['作物类型'].str.contains('蔬菜') & ~data['作物类型'].str.contains('豆类')]['作物名称'].unique().tolist()
mushroom_crops = data[data['作物类型'].str.contains('食用菌')]['作物名称'].unique().tolist()
```

```
# 获取豆类作物
grain_legume_crops = data[data['作物类型'].str.contains('粮食（豆类）')]['作物名称'].unique().tolist()
vegetable_legume_crops = data[data['作物类型'].str.contains('蔬菜（豆类）')]['作物名称'].unique().tolist()

# 计算每个作物的 2023 年销售量 (地块面积 × 亩产量)
data['2023 年销售量'] = data['地块面积/亩'] * data['亩产量/斤']

# 获取每个作物的 2023 年总销售量
demand_2023 = data.groupby('作物名称')['2023 年销售量'].sum().to_dict()

# 从数据集中提取作物的基础销售价格
base_price = data.set_index('作物名称')['平均销售单价'].to_dict()
# 确保 '作物名称' 和 '亩产量/斤' 列存在
base_yield = data.set_index('作物名称')['亩产量/斤'].to_dict()
# 从数据集中提取作物的基础种植成本
base_cost = data.set_index('作物名称')['种植成本/(元/亩)'].to_dict()

# 作物替代性矩阵
substitution_matrix = {
    '小麦': {'玉米': 0.8},
    '玉米': {'小麦': 0.8},
    # 其他作物替代关系
}
```

```
# 作物互补性矩阵
complementarity_matrix = {
    '豆类': {'小麦': 0.05},
    # 其他作物互补性关系
}

# 作物价格/成本相关性矩阵（可以自行定义相关性矩阵）
correlation_matrix = np.array([
    # 不同作物价格或成本的相关性
])

# 应用替代性
def apply_substitution(crop, current_price, current_cost):
    if crop in substitution_matrix:
        for substitute, factor in substitution_matrix[crop].items():
            substitute_price = base_price[substitute]
            substitute_cost = base_cost[substitute]
            if substitute_price / substitute_cost > current_price / current_cost:
                return substitute  # 返回替代作物
    return crop

# 应用互补性
def apply_complementarity(crop, year, field, solution):
    for past_year in range(year - 3, year):
        past_crop = solution.get((crop, field, past_year), None)
```

```
        if past_crop and past_crop in complementarity_matrix.get(crop, {}):
            return complementarity_matrix[crop][past_crop]  # 返回附加收益比例
    return 0

# 随机生成扰动值（考虑相关性）
def apply_correlation(crop, year):
    correlated_value = random.uniform(0.95, 1.05)  # 基于相关性随机扰动
    return correlated_value

# 检查种植方案是否符合季节性种植限制
def validate_seasonal_constraints(solution):
    for year in years:
        for field in fields:
            field_type = data[data['种植地块'] == field]['地块类型'].values[0]

            # 平旱地、梯田、山坡地：只能种植单季粮食类作物
            if field_type in ['平旱地', '梯田', '山坡地']:
                for crop in crops:
                    if crop not in grain_crops and solution.get((crop, field, year), 0) > 0:
                        return False  # 平旱地、梯田、山坡地只能种粮食类作物（水稻除外）

            # 水浇地：每年可以单季种植水稻或两季种植蔬菜
            if field_type == '水浇地':
                season1_crops = [crop for crop in vegetable_crops if crop not in ['大白菜', '白萝卜', '
红萝卜']]
                season2_crops = ['大白菜', '白萝卜', '红萝卜']
```

```
            # 如果第一季种了蔬菜（除大白菜、白萝卜、红萝卜外），第二季必须种大白菜、白萝卜或红萝卜之一
            if any(solution.get((crop, field, year), 0) > 0 for crop in season1_crops):
                if not any(solution.get((crop, field, year), 0) > 0 for crop in season2_crops):
                    return False  # 第二季必须种大白菜、白萝卜或红萝卜之一

            # 如果第二季种了大白菜、白萝卜或红萝卜，第一季必须种其他蔬菜
            if any(solution.get((crop, field, year), 0) > 0 for crop in season2_crops):
                if not any(solution.get((crop, field, year), 0) > 0 for crop in season1_crops):
                    return False  # 第一季必须种蔬菜

        # 普通大棚：第一季种蔬菜（除大白菜、白萝卜、红萝卜外），第二季种食用菌
        if field_type == '普通大棚':
            if not any(solution.get((crop, field, year), 0) > 0 for crop in vegetable_crops if crop not in ['大白菜', '白萝卜', '红萝卜']):
                return False  # 第一季必须种植蔬菜
            if not any(solution.get((crop, field, year), 0) > 0 for crop in mushroom_crops):
                return False  # 第二季必须种植食用菌

        # 智慧大棚：两季都可以种蔬菜，但第一季不能种大白菜、白萝卜、红萝卜
        if field_type == '智慧大棚':
            if any(solution.get((crop, field, year), 0) > 0 for crop in ['大白菜', '白萝卜', '红萝卜']):
                return False  # 第一季不允许种这些作物
```

```
    return True  # 如果没有违反约束，则返回 True
#计算利润
def calculate_profit(solution):
    total_profit = 0
    if not validate_seasonal_constraints(solution):
        return -1


    for year in years:
        for field in fields:
            for crop in crops:
                area = solution.get((crop, field, year), 0)
                if area == 0:
                    continue


                # 应用替代性
                crop = apply_substitution(crop, base_price[crop], base_cost[crop])


                # 销售量变化
                if crop in ['小麦', '玉米']:
                    growth_rate = random.uniform(0.05, 0.10)
                    demand = demand_2023[crop] * (1 + growth_rate) ** (year - 2023)
                else:
                    demand = demand_2023[crop] * random.uniform(0.95, 1.05)


                # 动态调整亩产量、种植成本、销售价格
                yield_per_acre = base_yield[crop] * random.uniform(0.9, 1.1)
```

```
                cost_per_acre = base_cost[crop] * (1 + 0.05) ** (year - 2023)
                price_per_unit = base_price[crop] * apply_correlation(crop, year)

                # 应用互补性
                complementarity_bonus = apply_complementarity(crop, year, field, solution)

                # 计算利润
                profit = (yield_per_acre * price_per_unit - cost_per_acre) * area * (1 + complementarity_bonus)
                total_profit += profit

    print(f'Profit: {total_profit}')  # 在此处打印每轮的收益值
    return total_profit


# 初始化解，生成初始种植方案
def initial_solution():
    solution = {}
    for year in years:
        for field in fields:
            field_area = float(data[data['种植地块'] == field]['地块面积/亩'].values[0])
            total_area = 0
            for crop in crops:
                if total_area >= field_area:
                    solution[(crop, field, year)] = 0
                else:
                    area = random.uniform(0, field_area - total_area)
```

```
                    solution[(crop, field, year)] = area
                    total_area += area
    return solution

# 确保豆类作物满足轮作约束
def ensure_legume_rotation(solution):
    legume_crops = data[data['作物类型'].str.contains('豆类')]['作物名称'].unique()
    for field in fields:
        for year in range(2024, 2031, 3):
            total_legume_area = sum(solution.get((crop, field, t), 0) for crop in legume_crops for t in range(year, min(year + 3, 2031)))
            if total_legume_area == 0:
                crop_to_adjust = random.choice(legume_crops)
                solution[(crop_to_adjust, field, random.choice(range(year, min(year + 3, 2031))))] = random.uniform(1, 3)
    return solution

# 邻域生成函数，生成解的邻域
def generate_neighbor(solution):
    neighbor = solution.copy()
    year = random.choice(list(years))
    field = random.choice(list(fields))
    crop = random.choice(list(crops))

    field_area = float(data[data['种植地块'] == field]['地块面积/亩'].values[0])
    total_area = sum(neighbor[(c, field, year)] for c in crops)
```

```
    adjustment = random.uniform(-5, 5)
    new_area = max(0, neighbor[(crop, field, year)] + adjustment)

    if total_area + adjustment <= field_area:
        neighbor[(crop, field, year)] = new_area
    else:
        neighbor[(crop, field, year)] = max(0, field_area - total_area)

    if not validate_seasonal_constraints(neighbor):
        return solution

    return neighbor

def simulated_annealing(initial_solution, max_iter=1000, initial_temp=1000, cooling_rate=0.99):
    current_solution = initial_solution()
    current_solution = ensure_legume_rotation(current_solution)
    current_profit = calculate_profit(current_solution)
    best_solution = current_solution
    best_profit = current_profit
    temperature = initial_temp

    # 保存每次迭代的收益值
    iter_profits = [current_profit]

    for i in range(max_iter):
        neighbor = generate_neighbor(current_solution)
        neighbor = ensure_legume_rotation(neighbor)
```

```
        neighbor_profit = calculate_profit(neighbor)

        # 更新解
        if neighbor_profit > current_profit or random.uniform(0, 1) < np.exp((neighbor_profit - current_profit) / temperature):
            current_solution = neighbor
            current_profit = neighbor_profit

        # 更新最佳解
        if current_profit > best_profit:
            best_solution = current_solution
            best_profit = current_profit

        # 保存当前收益值
        iter_profits.append(current_profit)

        # 降低温度
        temperature *= cooling_rate

    return best_solution, best_profit, iter_profits

# 运行模拟退火算法，并解包三个返回值
best_solution, best_profit, iter_profits = simulated_annealing(initial_solution)

# 将结果保存为 DataFrame
results = []
```

```
for (crop, field, year), area in best_solution.items():
    field_type = data[data['种植地块'] == field]['地块类型'].values[0]


    # 根据地块类型和作物类型来判断是第一季还是第二季
    if field_type in ['平旱地', '梯田', '山坡地']:
        season = '第一季'
        results.append([crop, field, year, area, season])
    elif field_type == '水浇地':
            # 水浇地第一季种蔬菜或水稻，第二季种大白菜、白萝卜和红萝卜
        if crop in ['大白菜', '白萝卜', '红萝卜']:
            season = '第二季'
        else:
            season = '第一季'
        results.append([crop, field, year, area, season])


        # 如果是蔬菜，第二季可以种植大白菜、白萝卜、红萝卜
        if crop in vegetable_crops and season == '第一季':
            for second_season_crop in ['大白菜', '白萝卜', '红萝卜']:
                results.append([second_season_crop, field, year, random.uniform(0.1, 0.5), '第二季'])


    elif field_type == '普通大棚':
        # 普通大棚第一季种蔬菜，第二季种食用菌
        if crop in mushroom_crops:
            season = '第二季'
        else:
```

```
            season = '第一季'
        results.append([crop, field, year, area, season])

        # 普通大棚第二季必须种食用菌
        if crop in vegetable_crops and season == '第一季':
            results.append(['食用菌', field, year, random.uniform(0.1, 0.5), '第二季'])

    elif field_type == '智慧大棚':
        # 智慧大棚两季都可以种蔬菜，但第一季不能种大白菜、白萝卜、红萝卜
        if crop in ['大白菜', '白萝卜', '红萝卜']:
            season = '第二季'
        else:
            season = '第一季'
        results.append([crop, field, year, area, season])

        # 智慧大棚可以种两季蔬菜
        if crop in vegetable_crops and season == '第一季':
            results.append([crop, field, year, random.uniform(0.1, 0.5), '第二季'])

# 创建 DataFrame 存储结果
df_results = pd.DataFrame(results, columns=['作物', '地块', '年份', '种植面积/亩', '季节'])

# 设定一个阈值过滤种植面积非常小的记录
threshold = 0.1
df_results['种植面积/亩'] = df_results['种植面积/亩'].apply(lambda x: 0 if abs(x) < threshold else x)
```

```
# 输出最终结果
print(df_results)

# 如果需要保存结果为文件
df_results.to_excel("optimal_solution_with_substitution_and_complementarity.xlsx", index=False)

# 计算每年的总收益
def calculate_annual_profit(df_results, data):
    # 合并模拟退火算法的结果和基础数据（包括亩产量、种植成本、销售单价）
    df_merged = pd.merge(df_results,
                         data[['作物名称', '种植地块', '亩产量/斤', '种植成本/(元/亩)', '平均销售单价']],
                         left_on=['作物', '地块'],
                         right_on=['作物名称', '种植地块'],
                         how='left')

    # 计算总收益 = (亩产量 * 销售单价 - 种植成本) * 种植面积
    df_merged['总收益'] = (df_merged['亩产量/斤'] * df_merged['平均销售单价'] - df_merged['种植成本/(元/亩)']) * df_merged['种植面积/亩']

    # 按年份汇总每年的总收益
    annual_profit = df_merged.groupby('年份')['总收益'].sum().reset_index()

    return annual_profit
```

```
# 调用函数，计算每年的总收益
annual_profit = calculate_annual_profit(df_results, cleaned_complete_data_df)

# 输出每年的总收益
print(annual_profit)

# 如果需要将结果保存到文件
annual_profit.to_excel("annual_profit_third_question.xlsx", index=False)

import matplotlib.pyplot as plt
import numpy as np

# 按年份和作物汇总种植面积
df_grouped = df_results.groupby(['年份', '作物'])['种植面积/亩'].sum().unstack()

# 设置图形大小
plt.figure(figsize=(14, 8))

# 定义年份和作物名称
years = df_grouped.index  # 获取年份
crops = df_grouped.columns  # 获取作物名称

# 设置 x 轴上的位置，以便所有年份的柱子能够并排显示
x = np.arange(len(crops))

# 定义柱子的宽度
```

```
width = 0.1


# 使用循环为每个年份绘制柱状图
for i, year in enumerate(years):
    plt.bar(x + i * width, df_grouped.loc[year], width=width, label=f'{year}年')


# 设置 x 轴和 y 轴的标签
plt.xlabel('作物名称')
plt.ylabel('种植面积/亩')
plt.title('2024~2030 年农作物的最优种植策略')


# 设置 x 轴上的刻度和作物名称
plt.xticks(x + width, crops, rotation=40)  # 作物名称旋转，避免重叠


# 添加网格和图例
plt.grid(True)
plt.legend()


# 显示图表
plt.tight_layout()
plt.show()
# 问题二和问题三的收益数据
years = [2024, 2025, 2026, 2027, 2028, 2029, 2030]
profits_question2 = [717288.290671, 874701.242564, 835609.674891, 728604.383763, 817417.159747, 685049.092080, 945241.288260]
```

```
profits_question3 = [867237.389527, 810132.681587, 874924.843207, 751834.863380, 879861.707575, 907530.505231, 872891.558839]


# 绘制收益对比图
plt.figure(figsize=(10, 6))


plt.plot(years, profits_question2, marker='o', linestyle='-', color='b', label='问题二-收益')
plt.plot(years, profits_question3, marker='o', linestyle='-', color='r', label='问题三-收益')


plt.title('问题二与问题三的经济效益对比图')
plt.xlabel('年份')
plt.ylabel('经济效益（总收益）')
plt.legend()
plt.grid(True)


# 显示图表
plt.show()
```