基于 MILP 和 GA 模型对农作物种植策略的研究

摘要

随着全球人口的增长和耕地资源的有限性，如何高效利用土地资源以实现农作物的可持续种植，已成为一个亟待解决的问题。在我国，充分利用有限的耕地资源、因地制宜发展有机种植产业，对乡村经济的可持续发展具有重要的现实意义。特别是在中国华北山区的乡村，由于其特殊的地理和气候条件，每年只能种植一季作物，这使得农作物的种植策略选择尤为重要。因此，必须提出优化的、合理的农作物种植方案，有效应对作物种植过程中的各种不确定性因素，从而提高作物产量和质量、降低农业生产的风险。

针对问题 1，我们将建立以收益最大化为目标的函数来寻找最优的农作物种植方案。根据题目要求与实际情况，我们将农作物分成单季作物和双季作物。对于单季作物，我们采用混合整数线性规划（MILP）模型进行目标函数最大化的求解；而对于双季作物，我们运用遗传算法（GA）来初始化 DEAP 框架，并实现适应度函数来评估解决方案的质量，通过执行进化算法来寻找最优解。基于附件中提供的农作物与耕地数据，我们分别得到了在滞销情况下与降价销售情况下的农作物种植策略，从而实现在不超过耕地资源的情况下实现收益最大化。

在问题 2 中，作物的预期销售量、亩产量、种植成本和销售价格都不确定，因此我们将采用随机规划方法来处理农作物种植中的不确定性问题。首先，根据历史数据和市场趋势，我们对不确定因素进行预测，将这些预测值作为模型的输入参数，并为它们分配适当的随机变量以模拟不确定性。在对数据进行动态调整后，我们运用蒙特卡洛模拟生成多种情景对不确定型和风险进行建模，并通过启发式算法遗传算法，得到在不确定性条件下表现最稳定、风险最小的种植方案。

问题 3 要求我们在问题 2 的基础上进一步考虑作物间的可替代性和互补性，以及销售量、价格和成本之间的相关性。为此，我们进一步引入了交叉弹性、互补性系数及相关性来构建一个更为复杂的多目标优化模型。由于约束条件与问题 2 相同，我们在问题 2 的基础上进一步分析了农作物的可替代性和互补性以及变量相关性对最优种植策略的影响。随后基于该结构方程模型构建风险模型，通过调整目标函数得到新的结果；在与问题 2 的结果对比中，我们可以确保所提出的种植策略在理论上是最优的。

关键词：农作物种植策略；线性规划模型；遗传算法；动态调整

一、问题重述

1.1 背景知识

随着全球人口的不断增长，对粮食和农产品的需求日益增加，而耕地资源有限，气候变化和环境问题也给农业生产带来了诸多挑战。在这样的背景下，如何科学合理地规划农作物的种植，提高土地的利用效率，增加农业产出，同时保障农业的可持续发展，成为了一个重要课题。

我国是一个农业大国，而华北山区作为中国重要的农业区域之一，具有独特的地理和气候条件，这些条件在一定程度上限制了农作物的种植种类和方式。由于该地区常年温度偏低，大多数耕地每年只能种植一季作物，因此必须充分利用有限的耕地资源，选择适宜的农作物种类、优化种植结构种植策略，从而实现农业生产效率的提高和农业发展风险的降低。

于是，本文利用题目给出的信息以及附件，通过建立数学模型帮助分析不同种植方案的经济效益和风险，优化资源配置，从而为华北山区的农作物种植提供最优策略。

1.2 具体问题

（1）假定各种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于 2023 年保持稳定，每季种植的农作物在当季销售。如果某种作物每季的总产量超过相应的预期销售量，超过部分不能正常销售。针对“超过部分滞销”和“超过部分按 2023 年销售价格的 50%降价销售”两种情况，分别给出该乡村 2024-2030 年农作物的最优种植方案。

（2）根据经验，小麦和玉米未来的预期销售量有增长的趋势，平均年增长率介于 5%-10%之间，其他农作物未来每年的预期销售量相对于 2023 年大约有±5%的变化。农作物的亩产量往往会受气候等因素的影响，每年会有±10%的变化。因受市场条件影响，农作物的种植成本平均每年增长 5%左右。粮食类作物的销售价格基本稳定；蔬菜类作物的销售价格有增长的趋势，平均每年增长 5%左右。食用菌的销售价格稳中有降，大约每年可下降 1%~5%，特别是羊肚菌的销售价格每年下降幅度为 5%。综合考虑各种农作物的预期销售量、亩产量、种植成本和销售价格的不确定性以及潜在的种植风险，给出该乡村 2024-2030 年农作物的最优种植方案。

（3）在现实生活中，各种农作物之间可能存在一定的可替代性和互补性，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。请在（2）的基础上综合考虑相关因素，给出该乡村 2024-2030 年农作物的最优种植策略，通过模拟数据进行求解，并与（2）的结果作比较分析。

二、问题分析

2.1 总体分析

华北山区作为中国重要的农业区域之一，寻找合理且高效的农作物种植策略是一个具有现实意义的课题。考虑到该地区特殊的地理和气候条件，以及耕地资源的有限性，本文的研究目标是最大化农作物的经济效益，同时降低种植风险。为此，本文将采用混合整数线性规划（MILP）和遗传算法（GA）两种方法，分别针对单季作物和双季作物的种植策略进行优化。

在总体分析中，我们首先识别了农作物种植过程中的关键因素，包括作物的预期销售量、种植成本、亩产量和销售价格。这些因素将直接影响到种植策略的制定和经济效益的评估。此外，我们还考虑了作物种植的约束条件，如耕地类型、作物生长周期、以及种植间隔要求等。通过对这些因素和约束的综合分析，我们能够为乡村制定出一套科学合理的种植策略。

2.2 具体分析

针对问题 1，我们将建立以收益最大化为目标的函数来寻找最优的农作物种植方案。根据题目要求与实际情况，我们将农作物分成单季作物和双季作物。对于单季作物，我们采用混合整数线性规划（MILP）模型进行目标函数最大化的求解；而对于双季作物，我们运用遗传算法（GA）来初始化 DEAP 框架，并实现适应度函数来评估解决方案的质量，通过执行进化算法来寻找最优解。

在问题 2 中，作物的预期销售量、亩产量、种植成本和销售价格都不确定，因此我们将采用随机规划方法来处理农作物种植中的不确定性问题。首先，根据历史数据和市场趋势，我们对不确定因素进行预测，将这些预测值作为模型的输入参数，并为它们分配适当的随机变量以模拟不确定性。在对数据进行动态调整后，我们运用蒙特卡洛模拟生成多种情景对不确定型和风险进行建模，并通过启发式算法遗传算法，从而得到在不确定性条件下表现最稳定、风险最小的种植方案。

问题 3 要求我们在问题 2 的基础上进一步考虑作物间的可替代性和互补性，以及销售量、价格和成本之间的相关性。为此，我们进一步引入了交叉弹性、互补性系数及相关性来构建一个更为复杂的多目标优化模型。由于约束条件与问题 2 相同，我们在问题 2 的基础上进一步分析了农作物的可替代性和互补性以及变量相关性对最优种植策略的影响。随后基于该结构方程模型构建风险模型，并将所得结果与问题 2 的结果进行对比分析。

三、模型假设

为更好地建立模型、解决问题，我们进行了以下假设：

1. 2024-2030 年农作物预期销售量、总种植成本、亩产量和销售价格相对于 2023 年保持稳定；

2. 每季种植的农作物在当季销售；

3. 2024-2030 年的预期销售量为 2023 年的实际产量。

四、符号说明

为了更好地描述本文所建立的模型，我们提出如下符号说明。

表 4.1 符号说明

符号	说明	单位
$x_{t,i}$	第 t 年种植第 i 中作物的面积	亩
$y_{t,i}$	第 t 年收获的第 i 种作物的总产量	斤
y_i^p	第 i 中作物的亩产量	斤
$z_{t,i}$	第 t 年销售的第 i 种作物的数量	斤
s_i	第 i 种作物的预期销售量	斤
$e_{t,i}$	第 t 年超过预期销售量的部分	斤

注：该表中所给为本题所建模型中的常用符号，运用次数较少的参量及其符号表示会在其所在节给出具体说明。

五、数据侧写

附件一包含了乡村的现有耕地和乡村种植的农作物的信息，附件二包含了 2023 年农作物种植情况与 2023 年统计的相关数据。由于两个附件所包含的信息息息相关，所以为更直观的呈现 2023 年乡村农作物的种植情况，我们首先对数据进行预处理，用 python 将附件一与附件二的文件进行合并，得到文件“附件 1+2”。

我们将所得数据分别按照地形和作物类型进行分类，得到了两个工作表“按地形分类”和“按作物分类”，其中，工作表“按作物分类（含总种植亩数）”中包含了每种作物的总种植面积。由于文件“附件 1+2”中并未包含 2023 年统计的相关数据，为进一步分析，我们将上面所得文件再次与“附件 2”进行连接，得到文件“连接结果”，并将结果分别按照地块类型和作物类型进行分类、汇总各种农作物的产量，得到工作表“2023 销售结果”。

由于附件中仅包含销售单价的区间，因此我们定义出销售价格的计算公式为

$$P_{\min} + ran \times (P_{\max} - P_{\min})$$

其中$P_{\min}$为价格最小值，$P_{\max}$为价格最大值，ran 为随机数；销量的计算公式则是（亩产量*种植面积）。除了得到各农作物的售价与销量，我们还必须探究每种作物适合种植的地块类型与季别。在对文件“附件 1+2”进行分析后，我们将每种作物所对应的种植地块进行总结，得到了文件“每种作物适合种植的地块类型与季别”与“每种作物适合种植的地块类型与季别_合并”。至此，数据预处理全部完成。

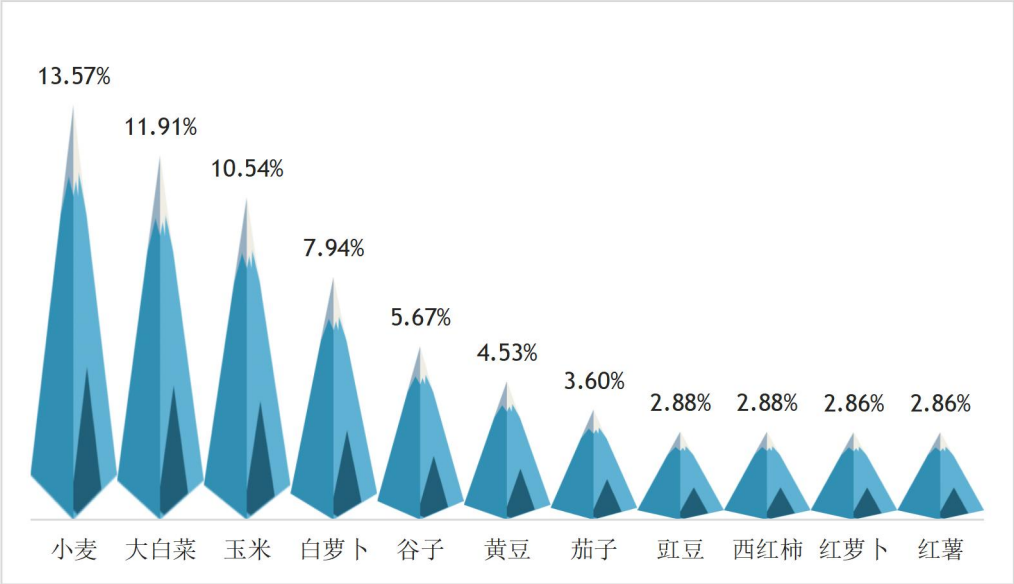

图 5.1 销量前十的作物及相应占比

由上图可知，在众多作物中，小麦、大白菜、玉米的销售量最大，销量分别为 170840 斤、150000 斤、132750 斤。且销量前五的作物中共有 2 种粮食，前十的作物中共有 6 种蔬菜，由此可以知道，华北山区对食材和粮食的市场需求量大，但蔬菜品类相对丰富，粮食选择则更集中在小麦、谷子等。因此，在选择最优种植策略时，应当更注重蔬菜品种的多样性，同时保证粮食作物的产量。

六、模型建立与求解

6.1 问题 1 的分析与求解

6.1.1 分析思路

根据题目条件，我们首先做出以下假设：

a. 每个地块最多种植两种作物，若该地块种植两种作物，则每种作物种植面积不得小于 40%；

b. 各种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于 2023 年保持稳定，且每季种植的农作物在当季销售；

c. 每种作物在同一地块不能连续重茬种植；

d. 每个地块的所有土地三年内至少种植一次豆类作物；每种作物每季的种植地不能太分散，若作物在多个同一类型的地块上种植，则地块编号必须连续。

有了以上假设约束，我们开始对最优种植策略进行求解。我们将建立以收益最大化为目标的函数来寻找最优的农作物种植方案。根据题目要求与实际情况，为方便讨论，我们将农作物分成单季作物和双季作物：单季作物是指只能在平旱地、梯田和山坡地种植的作物以及水稻，因为水稻只在水浇地单季种植；而双季作物则是指可以在水浇地、普通大棚和智慧大棚的作物。对于单季作物，我们采用混合整数线性规划（MILP）模型进行目标函数最大化的求解；而对于双季作物，我们运用遗传算法（GA），来初始化 DEAP 框架，即通过定义地块和作物的数据来实现适应度函数来评估解决方案的质量，通过执行进化算法来寻找最优解。基于附件中提供的农作物与耕地数据，我们分别得到在滞销情况下与降价销售情况下的农作物种植策略，从而实现在不超过耕地资源的情况下实现收益最大化。

图 6.1 问题 1 分析思路图

6.1.2 数据分析与处理

我们首先定义变量$x_{t,i}$，$y_{t,i}$，y_i^p，$z_{t,i}$，s_i，$e_{t,i}$，其中，

$$y_{t,i}=x_{t,i}\times y_i^p,\ z_{t,i}=\min(y_{t,i},s_i),\ e_{t,i}=\max(0,y_{t,i}-s_i)$$

定义了模型的基本变量，就可以根据题目要求的两种情况与单双季作物进行分类讨论。

在“超过部分滞销，造成浪费”的情况下，我们设定的利润最大化目标函数为

$$\max\Sigma(z_{t,i}\times p_i-x_{t,i}\times c_i)$$

其中，p_i是第 i 种作物的销售价格，c_i是第 i 种作物的种植成本。

而在“超过部分按 2023 年销售价格的 50%降价销售”情况下，我们设定的利润最大化目标函数为

$$\max\Sigma(z_{t,i}\times p_i+e_{t,i}\times 0.5\times p_i-x_{t,i}\times c_i)$$

对于单季作物而言，由于地块在一年内仅种植一种作物，求解的约束条件较少，因此我们直接运用混合整数线性规划模型即可得到最佳种植方案。

而对于双季作物，由于求解的约束条件较多，线性规划模型无法得出很好的方案，能够搜索、优化、学习并提供高质量解决方案的遗传算法（GA）成为了我

们解决问题的首选工具。通过定义地块和作物的数据，遗传算法首先对 DEAP 框架进行初始化，创建适合度类和个体类。接着用 evaluate 函数计算给定个体（即地块上种植的作物组合）的适应度值。同时，检查地块和作物的季节匹配情况、地块使用面积、作物产量是否超出预期销量、是否种植了豆类作物、是否存在连续重茬种植等问题。然后进行注册交叉（两点交叉，产生新的后代）、变异（均匀变异，改变个体中的某些基因）、选择（选择表现好的个体进入下一代）和评估（评估个体的适应度）操作。在遗传算法的主循环中，对于每一年初始化一个种群，并使用 DEAP 提供的 eaSimple 函数执行进化过程。进化过程中记录统计信息，并在每次迭代后选择最优个体。类比达尔文的进化理论，通过在进化中选择更适应环境的个体，并将其性状通过繁殖传给下一代，最终形成更多更具优势的后代。遗传算法便通过模仿自然选择和繁殖的过程，以寻找最优解。

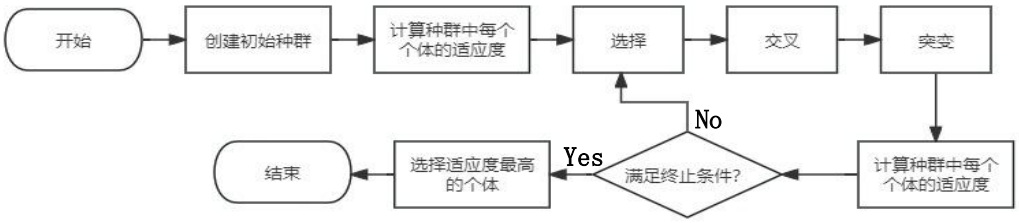

图 6.2 遗传算法分析思路图

6.2 问题 2 的分析与求解

6.2.1 分析思路

在问题 2 中，我们延续问题 1 求解中作出的假设，且仍然将作物分为单季作物和双季作物进行讨论；但由于作物的预期销售量、亩产量、种植成本和销售价格都不确定，因此我们将采用随机规划方法来处理农作物种植中的不确定性问题。

在数据预处理中，我们已经得到作物与地块的链接结果以及 2023 年的作物销售情况。根据历史数据和市场趋势，我们首先对不确定因素进行预测，将这些预测值作为模型的输入参数，并为它们分配适当的随机变量以模拟不确定性。在对数据进行动态调整后，我们运用蒙特卡洛模拟生成多种情景对不确定型和风险进行建模，并通过启发式算法遗传算法，得到在不确定性条件下表现最稳定、风险最小的种植方案。

6.2.2 数据分析与处理

在不确定的情况下，能使用随机数将求解问题与概率模型相结合的蒙特卡洛模拟就能很好得出问题近似解。我们通过 DEAP 框架实现遗传算法，并与蒙特卡洛模拟相结合，来预测和优化未来几年的农作物种植计划。我们设定的参数如下：群体大小为 100；遗传算法的终止进化代数为 250；交叉概率为 0.6；变异概率为 0.005。

我们定义了销售增长率、产量变化率、成本增长率和价格增长率，运用simulation函数来模拟未来几年的种植情况，包括销售量、亩产量、种植成本和销售价格的随机变化，同时生成了多种可能的种植情景。接下来使用遗传算法对DEAP框架进行初始化，创建适应度函数、个体、种群，并定义交叉、变异和选择操作。在满足一系列约束条件的基础上，我们使用evaluate函数计算个体的适应度，即总利润；又使用eaSimple算法运行遗传算法，对种植策略进行优化。

我们运用蒙特卡洛模拟生成未来几年的不确定性数据，如销售量、亩产量、成本和价格的随机变化，将这些数据用于评估不同种植策略在不同市场条件下的表现，从而帮助遗传算法找到最优或接近最优的种植策略。我们最终生成了一个最优的种植方案，该方案考虑了不确定性因素，并实现总利润最大化目标，这为决策者提供了一个基于数据和模型的种植策略建议。

6.3 问题3的分析与求解

6.3.1 分析思路

问题3要求我们在问题2的基础上进一步考虑作物间的可替代性和互补性，以及销售量、价格和成本之间的相关性。为此，我们进一步引入了交叉弹性、互补性系数及相关性来构建一个更为复杂的多目标优化模型。由于约束条件与问题2相同，我们在问题2的基础上进一步分析了农作物的可替代性和互补性以及变量相关性对最优种植策略的影响。随后基于该结构方程模型构建风险模型，通过调整目标函数得到新的结果，并与问题2中所得结果进行对比分析，便可确保我们所提出的种植策略在理论上是最优的。

6.3.2 数据分析与处理

(一)　作物之间的替代性与互补性

作物之间的替代性是指即一种作物可以部分替代另一种作物，这种关系可以用交叉弹性来衡量。交叉弹性是指一种作物的价格变化对另一种作物销售量的影响，即：

$$\text{交叉弹性}_{AB} = \frac{\%\Delta \text{需求量}_A}{\%\Delta \text{价格}_B}$$

若交叉弹性为正，表示作物A与作物B是替代品；若为负，则表示二者之间的关系是互补的。

互补性是指种植某种作物能够促进其他作物的生长或收益，例如，种植豆类作物可能会改良土壤，促进其他作物的生长。可以通过引入一个互补性系数来表示这种关系：

$$互补性系数_{AB}(0 \leqslant a_{AB} \leqslant 1)$$

如果作物 A 与 B 互补，则当二者在同一块地种植时，作物 A（B）的产量会相应增加。

(二)　销售量与价格、成本的相关性

由经济学原理可知，销售量与销售价格之间常存在负相关关系，即价格上升会导致需求下降，反之亦然。我们可以通过需求价格弹性来描述：

$$价格弹性 = \frac{\%\Delta 需求量}{\%\Delta 价格}$$

负的价格弹性意味着价格上升会导致销售量减少，反之亦然。

(三)　销售量与种植成本的相关性

市场条件可能导致某些作物的种植成本和销售量具有相关性。例如，成本增加可能导致销售量下降，反之成本下降可能促进销售量增加。这种相关性可以通过相关系数来描述：

$$相关系数_{成本-销售量}(-1 \leqslant pcs \leqslant 1)$$

正的相关系数意味着种植成本上升时，销售量也会上升；负相关则表示成本上升时销售量下降。

(四)　目标函数改进

在利润最大化的目标函数基础上，现需考虑作物替代性、互补性以及销售量、价格和成本的相关性。关于替代性，我们在目标函数中引入了一个调整系数：

$$x_{t,i} = x_{t,i} + \beta_{AB} \times \Delta 价格_B^t$$

其中，β_{AB} 是一个基于交叉弹性的系数，表示作物 B 的价格变化对作物 A 的种植决策的影响。

为了考虑销售量、价格和成本之间的相关性，我们还引入了一个相关性矩阵，用于调整这些变量之间的相互影响。具体来说就是当销售量、价格或成本发生变化时，可以通过相关性矩阵调整模型中的预期销售量或成本，从而得到调整后的最优种植策略。

七、模型评价与改进

7.1 模型评价

7.1.1 模型的优点

1. 实用性：模型基于华北山区某乡村的实际耕地和农作物数据构建，确保了模型的实用性和可操作性。

2. 灵活性：通过引入随机变量和蒙特卡洛模拟，模型能够灵活处理农作物

种植中的不确定性问题。

3. 优化性：模型采用混合整数线性规划和遗传算法，有效地寻找最优或近似最优的种植策略，以实现收益最大化或风险最小化。

4. 全面性：模型不仅考虑了农作物的经济效益，还考虑了种植约束条件和可替代性、互补性等因素，提供了全面的种植策略。

7.1.2 模型的缺点

1. 计算复杂性：随着模型中变量和约束条件的增加，模型的计算复杂性也随之增加，可能导致求解时间过长。

2. 现实适应性：模型可能未能完全考虑到实际农业生产中的所有因素，如自然灾害、政策变化等。

3. 动态调整限制：模型在处理动态调整方面存在一定局限性，特别是在作物种植结构调整和市场变化响应上。

7.2 模型改进

1. 算法优化：研究和应用更高效的优化算法，以减少模型的计算时间和提高求解速度。

2. 动态调整机制：引入动态调整机制，使模型能够根据市场变化和作物生长情况实时调整种植策略。

3. 多目标优化：进一步发展多目标优化模型，同时考虑经济效益、生态效益和社会效益，实现种植策略的综合优化。

4. 风险管理：引入风险评估和管理工具，如敏感性分析和压力测试，以评估和降低种植策略的风险。

八、参考文献

[1]CSDN.遗传算法（Genetic Algorithm）详解与实现[EB/OL](2024-07-16)遗传算法（Genetic Algorithm）详解与实现-CSDN博客

[2]CSDN.数学建模——蒙特卡罗算法（Monte Carlo Method）[EB/OL](2020-08-14)数学建模——蒙特卡罗算法（Monte Carlo Method）-CSDN博客

九、附录

9.1 数据预处理

9.1.1 按地形和作物分类

```
import pandas as pd
#合并附件1，附件2
df1 =r'D:\数模\C 题\附件 1.xlsx'
df_existing_fields = pd.read_excel(df1, sheet_name='乡村的现有耕地')
df2 =r'D:\数模\C 题\附件 2.xlsx'
df_crops_2023 = pd.read_excel(df2, sheet_name='2023 年的农作物种植情况')
merged_data = pd.merge(df_existing_fields, df_crops_2023,
                       left_on='地块名称', right_on='种植地块',
                       how='outer', suffixes=('_existing', '_crops'))

merged_data.fillna('', inplace=True)
merged_data.sort_values(by='地块名称', inplace=True)
columns_to_drop = ['种植地块']
merged_data.drop(columns=columns_to_drop, inplace=True, errors='ignore')
merged_data.to_excel('附件 1+2 结果.xlsx', index=False)
print("数据已分类并保存至'结果.xlsx'")

import pandas as pd
from openpyxl import Workbook, load_workbook
from openpyxl.utils.dataframe import dataframe_to_rows
import os
#分别按照地形和作物类型进行分类
file_path = r'D:\数模\C 题\数据预处理\附件 1+2 结果.xlsx'
if not os.path.exists(file_path):
    raise FileNotFoundError(f"文件 '{file_path}' 不存在，请检查路径是否正确。")
try:
    df = pd.read_excel(file_path)
    df_sorted = df.sort_values(by='作物名称')
    existing_wb = load_workbook(file_path)
    if "按作物分类" in existing_wb.sheetnames:
        del existing_wb["按作物分类"]
    new_ws = existing_wb.create_sheet(title="按作物分类")
    for r_idx, row in enumerate(dataframe_to_rows(df_sorted, index=False, header=True), 1):
        for c_idx, value in enumerate(row, 1):
            cell = new_ws.cell(row=r_idx, column=c_idx, value=value)
    existing_wb.save(file_path)
    print("数据已按作物名称分类，并保存至'结果.xlsx'的新 Sheet'按作物分类'中。")
except FileNotFoundError:
    print(f"文件 '{file_path}' 不存在，请检查路径是否正确。")
except Exception as e:
```

```
print(f"处理文件时发生错误：{e}")
```

9.1.2 链接“附件 1+2”与“附件 2”

```
import pandas as pd
file_path_existing = r'D:\数模\C 题\数据预处理\附件 1+2 结果.xlsx'
file_path_additional = r'D:\数模\C 题\附件 2.xlsx'
output_file_path = r'D:\数模\C 题\数据预处理\连接结果.xlsx'
df_existing = pd.read_excel(file_path_existing, sheet_name='Sheet1', engine='openpyxl')
df_additional = pd.read_excel(file_path_additional, sheet_name='2023 年统计的相关数据', engine='openpyxl')
merged_data = pd.merge(df_existing, df_additional, on='作物名称', how='outer', suffixes=('_existing', '_additional'))
merged_data.to_excel(output_file_path, index=False, engine='openpyxl')
print(f"数据已连接并保存至'{output_file_path}'。")
import pandas as pd
file_path = r'D:\数模\C 题\数据预处理\连接结果.xlsx'
output_file_path = r'D:\数模\C 题\数据预处理\连接结果.xlsx'
df = pd.read_excel(file_path, engine='openpyxl')
filtered_df = df[df['地块类型 B'] == df['地块类型 A']]
with pd.ExcelWriter(output_file_path, engine='openpyxl', mode='a') as writer:
    filtered_df.to_excel(writer, sheet_name='筛选结果（按照作物类型分类）', index=False)
print("数据已筛选并保存至'连接结果.xlsx'的新 Sheet'筛选结果（按照作物类型分类）'中。")
```

9.1.3 汇总亩产量

```
import pandas as pd
file_path = r'D:\数模\C 题\数据预处理\连接结果.xlsx'
output_file_path = r'D:\数模\C 题\数据预处理\连接结果.xlsx'
df = pd.read_excel(file_path, sheet_name='2023 销售结果', engine='openpyxl')
# 按照“作物编号”分组并计算“亩产量/斤”的总和
grouped_df = df.groupby('作物编号')['亩产量/斤'].sum().reset_index()
with pd.ExcelWriter(output_file_path, engine='openpyxl', mode='a', if_sheet_exists='replace') as writer:
    grouped_df.to_excel(writer, sheet_name='亩产量汇总', index=False)
print("数据已按作物编号汇总，并保存至'连接结果.xlsx'的新 Sheet'亩产量汇总'中。")
```

9.1.4 计算销量、销售价格

```
import pandas as pd
import numpy as np
file_path = r'D:\数模\C 题\数据预处理\连接结果.xlsx'
output_file_path = r'D:\数模\C 题\数据预处理\连接结果.xlsx'
df_sales = pd.read_excel(file_path, sheet_name='2023 销售结果', engine='openpyxl')
# 计算“亩产量/斤”与“种植面积/亩”的乘积
df_sales['作物产量（销量）'] = df_sales['亩产量/斤'] * df_sales['种植面积/亩']
# 处理“销售单价/(元/斤)”列
df_sales[['最小值', '最大值']] = df_sales['销售单价/(元/斤)'].str.split('-', expand=True).astype(float)
df_sales['价格区间'] = df_sales['最大值'] - df_sales['最小值']
# 选择需要的列
selected_columns = [
    '作物编号',
```

```
        '作物名称',
        '亩产量/斤',
        '种植面积/亩',
        '作物产量（销量）',
        '销售单价/(元/斤)',
        '最小值',
        '最大值',
        '价格区间'
    ]
    result_df = df_sales[selected_columns]
    with pd.ExcelWriter(output_file_path, engine='openpyxl', mode='a', if_sheet_exists='replace') as writer:
        result_df.to_excel(writer, sheet_name='销售结果分析', index=False)
    print(f"数据已处理并保存至'{output_file_path}'的新 Sheet'销售结果分析'中。")


    file_path = r'D:\华工\数模\C 题\数据预处理\连接结果.xlsx'
    df_sales = pd.read_excel(file_path, sheet_name='销售结果分析', engine='openpyxl')
    # 生成实际销售价格
    np.random.seed(42)  # 设置随机种子以获得可重复的结果
    df_sales['实际销售价格'] = df_sales['最小值'] + (np.random.rand(len(df_sales)) * df_sales['价格区间'])
    with pd.ExcelWriter(file_path, engine='openpyxl', mode='a', if_sheet_exists='replace') as writer:
        df_sales.to_excel(writer, sheet_name='销售结果分析', index=False)
    print(f"数据已处理并保存至'{file_path}'的 Sheet'销售结果分析'中。")
```

9.1.5 匹配农作物与地块

```
import pandas as pd
import numpy as np
file_path_input = r'D:\数模\C 题\数据预处理\种植情况.xlsx'
file_path_output = r'D:\数模\C 题\数据预处理\每种作物适合种植的地块类型与季别.xlsx'
df_planting = pd.read_excel(file_path_input, sheet_name='种植情况', engine='openpyxl')
new_columns = [
    '平旱地', '梯田', '山坡地',
    '水浇地单季', '水浇地第一季', '普通大棚第一季', '智慧大棚第一季',
    '水浇地第二季', '普通大棚第二季', '智慧大棚第二季'
]
for col in new_columns:
    df_planting[col] = 0
for index, row in df_planting.iterrows():
    if row['地块类型'] == '平旱地':
        df_planting.at[index, '平旱地'] = 1
    elif row['地块类型'] == '梯田':
        df_planting.at[index, '梯田'] = 1
    elif row['地块类型'] == '山坡地':
        df_planting.at[index, '山坡地'] = 1
    elif row['地块类型'] == '水浇地':
        if row['种植季次'] == '单季':
```

```
                df_planting.at[index, '水浇地单季'] = 1
            elif row['种植季次'] == '第一季':
                df_planting.at[index, '水浇地第一季'] = 1
            elif row['种植季次'] == '第二季':
                df_planting.at[index, '水浇地第二季'] = 1
        elif row['地块类型'] == '普通大棚':
            if row['种植季次'] == '第一季':
                df_planting.at[index, '普通大棚第一季'] = 1
            elif row['种植季次'] == '第二季':
                df_planting.at[index, '普通大棚第二季'] = 1
        elif row['地块类型'] == '智慧大棚':
            if row['种植季次'] == '第一季':
                df_planting.at[index, '智慧大棚第一季'] = 1
            elif row['种植季次'] == '第二季':
                df_planting.at[index, '智慧大棚第二季'] = 1
    df_sorted = df_planting.sort_values(by='作物编号')
    selected_columns = [
        '作物编号', '作物名称', '作物类型'
    ] + new_columns
    result_df = df_sorted[selected_columns]
    result_df.to_excel(file_path_output, index=False, engine='openpyxl')
    print(f"数据已处理并保存至'{file_path_output}'。")

    import pandas as pd
    file_path_input = r'D:\数模\C 题\数据预处理\每种作物适合种植的地块类型与季别.xlsx'
    file_path_output = r'D:\数模\C 题\数据预处理\每种作物适合种植的地块类型与季别_合并.xlsx'
    df_planting = pd.read_excel(file_path_input, sheet_name='每种作物适合种植的地块类型与季别', engine='openpyxl')
    grouped_df = df_planting.groupby(['作物编号', '作物名称', '作物类型']).max().reset_index()
    grouped_df.to_excel(file_path_output, index=False, engine='openpyxl')
    print(f"数据已合并并保存至'{file_path_output}'。")
```

9.2 问题 1

9.2.1 超过部分滞销，造成浪费

（一）单季作物

```
import pandas as pd
from ortools.linear_solver import pywraplp

# 文件路径
crop_types_path = r'D:\数模\C 题\数据预处理\第一题第一问\作物种类.xlsx'
sales_results_path = r'D:\数模\C 题\数据预处理\第一题第一问\连接结果.xlsx'
analysis_path = r'D:\数模\C 题\数据预处理\第一题第一问\连接结果.xlsx'
suitable_plots_path = r'D:\数模\C 题\数据预处理\第一题第一问\每种作物适合种植的地块类型与季别_合并.xlsx'
output_path = r'D:\数模\C 题\数据预处理\第一题第一问\第一季单季作物.xlsx'

```

```
    # 读取 Excel 文件
    crop_types_df = pd.read_excel(crop_types_path, sheet_name='第一季（单）')
    sales_results_df = pd.read_excel(sales_results_path, sheet_name='2023 销售结果')
    analysis_df = pd.read_excel(analysis_path, sheet_name='销售结果分析')
    suitable_plots_df = pd.read_excel(suitable_plots_path)

    # 获取地块信息
    plots = {
        '平旱地': ['A1', 'A2', 'A3', 'A4', 'A5', 'A6'],
        '梯田': ['B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B9', 'B10', 'B11', 'B12', 'B13', 'B14'],
        '山坡地': ['C1', 'C2', 'C3', 'C4', 'C5', 'C6']
    }
    areas = {
        'A1': 80, 'A2': 55, 'A3': 35, 'A4': 72, 'A5': 68, 'A6': 55,
        'B1': 60, 'B2': 46, 'B3': 40, 'B4': 28, 'B5': 25,
        'B6': 86, 'B7': 55, 'B8': 44, 'B9': 50, 'B10': 25,
        'B11': 60, 'B12': 45, 'B13': 35, 'B14': 20,
        'C1': 15, 'C2': 13, 'C3': 15, 'C4': 18, 'C5': 27, 'C6': 20
    }

    # 提取作物相关数据
    crops = list(set(crop_types_df['作物名称']))
    crop_info = {}
    for index, row in analysis_df.iterrows():
        crop_info[row['作物名称']] = {
            '产量': row['作物产量（销量）'],
            '价格': row['实际销售价格'],
            '成本': row['种植成本/(元/亩)'],
            '地块': row['地块名称'],
            '地块类型': row['地块类型']
        }
        # 标记豆类作物
        if '豆类' in row['作物类型']:
            crop_info[row['作物名称']]['is_bean'] = True
        else:
            crop_info[row['作物名称']]['is_bean'] = False
    # 创建线性规划模型
    solver = pywraplp.Solver.CreateSolver('SCIP')

    # 定义变量
    variables = {}
    for year in range(2024, 2031):
        for plot_type, plots_list in plots.items():
            for plot in plots_list:
```

```
                for crop in crops:
                    # 使用布尔索引
                    if len(suitable_plots_df.loc[suitable_plots_df['作物名称'] == crop, plot_type]) > 0 and \
                       suitable_plots_df.loc[suitable_plots_df['作物名称'] == crop, plot_type].iloc[0] == 1:
                        variables[(year, plot, crop)] = solver.NumVar(0, areas[plot], f'x_{year}_{plot}_{crop}')

    binary_vars = {}

    for year in range(2024, 2031):
        for plot_type, plots_list in plots.items():
            for plot in plots_list:
                for crop in crops:
                    # 使用布尔索引
                    if len(suitable_plots_df.loc[suitable_plots_df['作物名称'] == crop, plot_type]) > 0 and \
                       suitable_plots_df.loc[suitable_plots_df['作物名称'] == crop, plot_type].iloc[0] == 1:
                        variables[(year, plot, crop)] = solver.NumVar(0, areas[plot], f'x_{year}_{plot}_{crop}')
                        binary_vars[(year, plot, crop)] = solver.BoolVar(f'y_{year}_{plot}_{crop}')


    # 添加约束条件
    # 每个地块总种植面积不得超过该地块面积
    for year in range(2024, 2031):
        for plot in areas.keys():
            solver.Add(sum(variables.get((year, plot, crop), 0) for crop in crops) <= areas[plot])

    # 每种作物在同一地块不能连续重茬种植
    for year in range(2024, 2030):  # 不考虑最后一年的重茬
        for plot in areas.keys():
            for crop in crops:
                solver.Add(variables.get((year, plot, crop), 0) + variables.get((year+1, plot, crop), 0) <= areas[plot])

    # 每个地块的所有土地三年内至少种植一次豆类作物
    for plot in areas.keys():
        solver.Add(sum(variables.get((year, plot, crop), 0) * crop_info[crop]['is_bean'] for year in range(2024, 2027) for crop in crops if '
is_bean' in crop_info[crop]) >= 1)

    # 每种作物每季的种植地不能太分散
    # 假设作物连续地块为 A1 到 A6
    #for year in range(2024, 2031):
    #   for crop in crops:
    #       for plot_type, plots_list in plots.items():
    #           for i in range(len(plots_list) - 1):
    #               solver.Add(sum(variables.get((year, plots_list[i], crop), 0)) + sum(variables.get((year, plots_list[i+1], crop), 0)) <=
areas[plots_list[i]])
```

```
# 每种作物每季的种植地必须连续
for year in range(2024, 2031):
    for crop in crops:
        for plot_type, plots_list in plots.items():
            for i in range(len(plots_list) - 1):
                # 如果作物在当前地块和下一个地块上都有种植，则它们的总面积不得超过当前地块的面积
                solver.Add(
                    variables.get((year, plots_list[i], crop), 0) +
                    variables.get((year, plots_list[i+1], crop), 0) <=
                    areas[plots_list[i]]
                )

# 设置目标函数
objective = solver.Objective()
for year in range(2024, 2031):
    for plot in areas.keys():
        for crop in crops:
            if crop in crop_info:
                profit_per_acre = (crop_info[crop]['价格'] * crop_info[crop]['产量']) - crop_info[crop]['成本']
                var = variables.get((year, plot, crop))
                if var is not None:
                    objective.SetCoefficient(var, profit_per_acre)

objective.SetMaximization()

# 求解模型
status = solver.Solve()

results = []
if status == pywraplp.Solver.OPTIMAL:
    print('Optimal solution found.')
    # 输出最优解
    for key, var in variables.items():
        if var.solution_value() > 0:
            results.append({
                '年份': key[0],
                '地块': key[1],
                '作物': key[2],
                '种植面积': var.solution_value()
            })
else:
    print('No optimal solution found.')

# 将结果保存到Excel文件
```

```
results_df = pd.DataFrame(results)
with pd.ExcelWriter(output_path) as writer:
    results_df.to_excel(writer, sheet_name='第一季单季作物', index=False)

print(f'Results saved to {output_path}')
```

（二）双季作物

```
import pandas as pd
from deap import base, creator, tools, algorithms
import numpy as np
from random import randint, choice
from openpyxl import Workbook
from openpyxl.utils.dataframe import dataframe_to_rows
import os

# 数据路径
planting_restrictions_path = r'D:\数模\C 题\第一题\第二问\双季作物\使用数据\作物种植限制.xlsx'
sales_results_path = r'D:\数模\C 题\数据预处理\第一题第一问\连接结果（双季）.xlsx'
suitability_path = r'D:\数模\C 题\数据预处理\第一题第一问\每种作物适合种植的地块类型与季别_合并.xlsx'

# 加载数据
planting_restrictions_df = pd.read_excel(planting_restrictions_path)
sales_results_df = pd.read_excel(sales_results_path, sheet_name='2023 销售结果')

# 地块信息
land_info = {
    "水浇地第一季": {"areas": [15, 10, 14, 6, 10, 12, 22, 20], "names": ["D1", "D2", "D3", "D4", "D5", "D6", "D7", "D8"]},
    "水浇地第二季": {"areas": [15, 10, 14, 6, 10, 12, 22, 20], "names": ["D1", "D2", "D3", "D4", "D5", "D6", "D7", "D8"]},
    "普通大棚第一季": {"areas": [0.6] * 16, "names": ["E{}".format(i+1) for i in range(16)]},
    "普通大棚第二季": {"areas": [0.6] * 16, "names": ["E{}".format(i+1) for i in range(16)]},
    "智慧大棚第一季": {"areas": [0.6] * 4, "names": ["F{}".format(i+1) for i in range(4)]},
    "智慧大棚第二季": {"areas": [0.6] * 4, "names": ["F{}".format(i+1) for i in range(4)]}
}

# 获取每种地块类型允许种植的作物列表
crop_info = {}
for plot_type in land_info.keys():
    allowed_crops = planting_restrictions_df[planting_restrictions_df['地块类型'] == plot_type][['作物名称', '作物类型']]
    crop_info[plot_type] = allowed_crops.to_dict(orient='records')

# 销售信息
sales_info = {}
for _, row in sales_results_df.iterrows():
     sales_info[row['作物名称']] = {'expected_sales': row['作物产量（销量）'], 'price': row['实际销售价格'], 'cost': row['种植成本/(元/亩)'], 'discount_price': row['实际销售价格'] * 0.5}
```

```
38.
39.         # 初始化 DEAP 框架
40.         creator.create("FitnessMax", base.Fitness, weights=(1.0,))
41.         creator.create("Individual", list, fitness=creator.FitnessMax)
42.
43.         toolbox = base.Toolbox()
44.         toolbox.register("attr_int", lambda: randint(0, len(crop_info[list(crop_info.keys())[0]])-1) if len(crop_info[list(crop_info.keys())[0]])
    > 0 else -1)
45.         toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_int, len(land_info.keys()) * max(len(v['areas']) for v
    in land_info.values()))
46.
47.         toolbox.register("population", tools.initRepeat, list, toolbox.individual)
48.
49.         # 适应度函数
50.         def evaluate(individual):
51.             fitness = 0
52.             plot_usage = {plot: 0 for plot in land_info.keys()}
53.             plot_crops = {plot: [] for plot in land_info.keys()}
54.             crop_production = {crop['作物名称']: 0 for plot in land_info.keys() for crop in crop_info[plot]}
55.             plots_used = {plot: False for plot in land_info.keys()}
56.
57.             index = 0
58.             for plot, info in land_info.items():
59.                 areas = info['areas']
60.                 names = info['names']
61.                 plot_type = plot
62.                 for i, name in enumerate(names):
63.                     crop_index = individual[index]
64.                     index += 1
65.                     if crop_index >= 0 and crop_index < len(crop_info[plot_type]):
66.                         crop = crop_info[plot_type][crop_index]
67.                         crop_name = crop['作物名称']
68.                         plot_area = areas[i % len(areas)]
69.
70.                         # 检查地块的使用面积
71.                         if plot_usage[plot] + plot_area > sum(areas):
72.                             return -1,
73.
74.                         plot_usage[plot] += plot_area
75.                         plot_crops[plot].append((crop_name, plot_area))
76.                         crop_production[crop_name] += plot_area
77.
78.                         # 检查作物是否有销售信息
79.                         if crop_name not in sales_info:
```

```
                    continue

                expected_sales = sales_info[crop_name]['expected_sales']

                # 如果产量超过了预期销售量，则超出部分视为滞销
                if crop_production[crop_name] > expected_sales:
                    regular_sales = expected_sales
                    excess_production = crop_production[crop_name] - expected_sales
                    crop_production[crop_name] = regular_sales + excess_production

                    # 计算正常销售部分的利润
                    regular_profit = regular_sales * (sales_info[crop_name]['price'] - sales_info[crop_name]['cost'])

                    # 超出部分不计入利润
                    fitness += regular_profit
                else:
                    # 计算利润
                    profit = crop_production[crop_name] * (sales_info[crop_name]['price'] - sales_info[crop_name]['cost'])
                    fitness += profit
                plots_used[plot] = True


        # 检查豆类作物种植
        for plot in plots_used:
            if not plots_used[plot]:
                continue

            crops_in_plot = [crop for crop, _ in plot_crops[plot]]
            if not any('豆类' in crop['作物类型'] for crop in crop_info[plot]):
                return -1,

        # 检查连续重茬种植
        for plot in plots_used:
            crops_in_plot = [crop for crop, _ in plot_crops[plot]]
            if len(set(crops_in_plot)) < len(crops_in_plot):
                return -1,

        return fitness,

    # 交叉操作
    toolbox.register("mate", tools.cxTwoPoint)
    # 变异操作
    toolbox.register("mutate", tools.mutUniformInt, low=0, up=max(len(v) for v in crop_info.values()), indpb=0.05)
    # 选择操作
```

```
toolbox.register("select", tools.selTournament, tournsize=3)
# 评价操作
toolbox.register("evaluate", evaluate)


# 主函数
def main():
    years = range(2024, 2031)

    for year in years:
        # 初始化种群
        pop = toolbox.population(n=50)

        # 进化过程
        stats = tools.Statistics(lambda ind: ind.fitness.values)
        stats.register("avg", np.mean)
        stats.register("std", np.std)
        stats.register("min", np.min)
        stats.register("max", np.max)

        pop, logbook = algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=20, stats=stats, verbose=True)

        # 找出最佳个体
        best_ind = tools.selBest(pop, 1)[0]

        # 将结果保存到Excel
        save_results_to_excel(best_ind, year)

def save_results_to_excel(individual, year):
    results = []
    index = 0
    for plot, info in land_info.items():
        areas = info['areas']
        names = info['names']
        for i, name in enumerate(names):
            crop_index = individual[index]
            index += 1
            if crop_index >= 0 and crop_index < len(crop_info[plot]):
                crop = crop_info[plot][crop_index]
                plot_area = areas[i % len(areas)]
                results.append({
                    '年份': year,
                    '地块类型': plot,
                    '地块名称': name,
                    '种植的作物名称': crop['作物名称'],
```

```
                    '种植作物的面积': plot_area
                })

    # 创建Excel文件
    filename = f'results_{year}.xlsx'
    if os.path.exists(filename):
        mode = 'a'  # Append mode if the file exists
    else:
        mode = 'w'  # Write mode if the file does not exist

    wb = Workbook()
    ws = wb.active
    ws.append(['年份', '地块类型', '地块名称', '种植的作物名称', '种植作物的面积'])

    for result in results:
        ws.append([
            result['年份'],
            result['地块类型'],
            result['地块名称'],
            result['种植的作物名称'],
            result['种植作物的面积']
        ])

    try:
        wb.save(filename)
    except PermissionError:
        raise PermissionError(f"Could not write to '{filename}'. Please close any open instances of this file.")

if __name__ == "__main__":
    main()
```

9.2.2 超过部分按2023年销售价格的50%降价出售

（一）单季作物

```
import pandas as pd
from ortools.linear_solver import pywraplp

# 文件路径
paths = {
    'crop_types': r'D:\数模\C题\数据预处理\第一题第一问\第一题第一问(dan)\第一题第一问 - 副本\作物种类.xlsx',
    'sales_results': r'D:\数模\C题\数据预处理\第一题第一问\第一题第一问(dan)\第一题第一问 - 副本\连接结果.xlsx',
    'analysis': r'D:\数模\C题\数据预处理\第一题第一问\第一题第一问(dan)\第一题第一问 - 副本\连接结果.xlsx',
    'suitable_plots': r'D:\数模\C题\数据预处理\第一题第一问\第一题第一问(dan)\第一题第一问 - 副本\每种作物适合种植的地块类型与季别_合并.xlsx',
    'output': r'D:\数模\C题\数据预处理\第一题第一问\第一季单季作物.xlsx'
}

```

```python
# 读取 Excel 文件
crop_types_df = pd.read_excel(paths['crop_types'], sheet_name='第一季（单）')
sales_results_df = pd.read_excel(paths['sales_results'], sheet_name='2023 销售结果')
analysis_df = pd.read_excel(paths['analysis'], sheet_name='销售结果分析')
suitable_plots_df = pd.read_excel(paths['suitable_plots'])

# 获取地块信息
plots = {
    '平旱地': ['A1', 'A2', 'A3', 'A4', 'A5', 'A6'],
    '梯田': ['B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B9', 'B10', 'B11', 'B12', 'B13', 'B14'],
    '山坡地': ['C1', 'C2', 'C3', 'C4', 'C5', 'C6']
}
areas = {
    'A1': 80, 'A2': 55, 'A3': 35, 'A4': 72, 'A5': 68, 'A6': 55,
    'B1': 60, 'B2': 46, 'B3': 40, 'B4': 28, 'B5': 25,
    'B6': 86, 'B7': 55, 'B8': 44, 'B9': 50, 'B10': 25,
    'B11': 60, 'B12': 45, 'B13': 35, 'B14': 20,
    'C1': 15, 'C2': 13, 'C3': 15, 'C4': 18, 'C5': 27, 'C6': 20
}

# 提取作物相关数据
crops = list(set(crop_types_df['作物名称']))
crop_info = {}
for index, row in analysis_df.iterrows():
    crop_info[row['作物名称']] = {
        '产量': row['作物产量（销量）'],
        '价格': row['实际销售价格'],
        '成本': row['种植成本/(元/亩)'],
        '地块': row['地块名称'],
        '地块类型': row['地块类型']
    }
    # 标记豆类作物
    if '豆类' in row['作物类型']:
        crop_info[row['作物名称']]['is_bean'] = True
    else:
        crop_info[row['作物名称']]['is_bean'] = False

# 创建线性规划模型
solver = pywraplp.Solver.CreateSolver('SCIP')

# 定义变量
variables = {}
for year in range(2024, 2031):
    for plot_type, plots_list in plots.items():
```

```
            for plot in plots_list:
                for crop in crops:
                    if len(suitable_plots_df.loc[suitable_plots_df['作物名称'] == crop, plot_type]) > 0 and \
                       suitable_plots_df.loc[suitable_plots_df['作物名称'] == crop, plot_type].iloc[0] == 1:
                        variables[(year, plot, crop)] = solver.NumVar(0, areas[plot], f'x_{year}_{plot}_{crop}')

binary_vars = {}

for year in range(2024, 2031):
    for plot_type, plots_list in plots.items():
        for plot in plots_list:
            for crop in crops:
                # 使用布尔索引
                if len(suitable_plots_df.loc[suitable_plots_df['作物名称'] == crop, plot_type]) > 0 and \
                   suitable_plots_df.loc[suitable_plots_df['作物名称'] == crop, plot_type].iloc[0] == 1:
                    variables[(year, plot, crop)] = solver.NumVar(0, areas[plot], f'x_{year}_{plot}_{crop}')
                    binary_vars[(year, plot, crop)] = solver.BoolVar(f'y_{year}_{plot}_{crop}')


# 添加约束条件
# 每个地块总种植面积不得超过该地块面积
for year in range(2024, 2031):
    for plot in areas.keys():
        solver.Add(sum(variables.get((year, plot, crop), 0) for crop in crops) <= areas[plot])

#若该地块种植多种作物，则每种作物种植面积不得小于该地块面积的10%
#for year in range(2024, 2031):
#   for plot in areas.keys():
#       for crop in crops:
#         solver.Add(variables.get((year, plot, crop), 0) >= 5)
#         solver.Add(variables.get((year, plot, crop), 0) >= 0.0001 * areas[plot])

# 若该地块种植多种作物，则每种作物种植面积不得小于该地块面积的10%
# 使用二进制变量来表示地块是否被使用
for year in range(2024, 2031):
    for plot in areas.keys():
        for crop in crops:
            if (year, plot, crop) in variables:
                solver.Add(variables.get((year, plot, crop), 0) >= 0.1 * areas[plot] * binary_vars.get((year, plot, crop), 0))

# 每种作物在同一地块不能连续重茬种植
for year in range(2024, 2030):  # 不考虑最后一年的重茬
    for plot in areas.keys():
        for crop in crops:
```

```
                solver.Add(variables.get((year, plot, crop), 0) + variables.get((year+1, plot, crop), 0) <= areas[plot])

    # 每个地块的所有土地三年内至少种植一次豆类作物
    for plot in areas.keys():
        solver.Add(sum(variables.get((year, plot, crop), 0) * crop_info[crop]['is_bean'] for year in range(2024, 2027) for crop in crops if '
    is_bean' in crop_info[crop]) >= 1)

    # 每种作物每季的种植地必须连续
    for year in range(2024, 2031):
        for crop in crops:
            for plot_type, plots_list in plots.items():
                for i in range(len(plots_list) - 1):
                    # 如果作物在当前地块和下一个地块上都有种植，则它们的总面积不得超过当前地块的面积
                    solver.Add(
                        variables.get((year, plots_list[i], crop), 0) +
                        variables.get((year, plots_list[i+1], crop), 0) <=
                        areas[plots_list[i]]
                    )


    # 设置目标函数
    objective = solver.Objective()
    for year in range(2024, 2031):
        for plot in areas.keys():
            for crop in crops:
                if crop in crop_info:
                    profit_per_acre = (crop_info[crop]['价格'] * crop_info[crop]['产量']) - crop_info[crop]['成本']
                    var = variables.get((year, plot, crop))
                    if var is not None:
                        objective.SetCoefficient(var, profit_per_acre)

    objective.SetMaximization()

    # 求解模型
    status = solver.Solve()

    results = []
    excess_results = []
    if status == pywraplp.Solver.OPTIMAL:
        print('Optimal solution found.')
        # 输出最优解
        for key, var in variables.items():
            if var.solution_value() > 0:
                results.append({
```

```
                '年份': key[0],
                '地块': key[1],
                '作物': key[2],
                '种植面积': var.solution_value()
            })

    # 计算超出部分的收益
    for crop in crops:
        for year in range(2024, 2031):
            total_production = sum([var.solution_value() for key, var in variables.items() if key[0] == year and key[2] == crop])
            expected_sales = crop_info[crop]['产量']
            if total_production > expected_sales:
                excess = total_production - expected_sales
                excess_profit = (crop_info[crop]['价格'] * 0.5 - crop_info[crop]['成本']) * excess
                excess_results.append({
                    '年份': year,
                    '作物': crop,
                    '超出产量': excess,
                    '超出收益': excess_profit
                })

else:
    print('No optimal solution found.')

# 将结果保存到 Excel 文件
results_df = pd.DataFrame(results)
with pd.ExcelWriter(paths['output']) as writer:
    results_df.to_excel(writer, sheet_name='第一季单季作物', index=False)
    if excess_results:
        pd.DataFrame(excess_results).to_excel(writer, sheet_name='超出部分收益', index=False)

print(f'Results saved to {paths["output"]}')
```

（二）双季作物

```
import pandas as pd
from deap import base, creator, tools, algorithms
import numpy as np
from random import randint, choice
from openpyxl import Workbook
from openpyxl.utils.dataframe import dataframe_to_rows
import os

# 数据路径
planting_restrictions_path = r'D:\数模\C 题\第一题\第二问\双季作物\使用数据\作物种植限制.xlsx'
sales_results_path = r'D:\数模\C 题\数据预处理\第一题第一问\连接结果（双季）.xlsx'
```

```
    suitability_path = r'D:\数模\C 题\数据预处理\第一题第一问\每种作物适合种植的地块类型与季别_合并.xlsx'

    # 加载数据
    planting_restrictions_df = pd.read_excel(planting_restrictions_path)
    sales_results_df = pd.read_excel(sales_results_path, sheet_name='2023 销售结果')

    # 地块信息
    land_info = {
        "水浇地第一季": {"areas": [15, 10, 14, 6, 10, 12, 22, 20], "names": ["D1", "D2", "D3", "D4", "D5", "D6", "D7", "D8"]},
        "水浇地第二季": {"areas": [15, 10, 14, 6, 10, 12, 22, 20], "names": ["D1", "D2", "D3", "D4", "D5", "D6", "D7", "D8"]},
        "普通大棚第一季": {"areas": [0.6] * 16, "names": ["E{}".format(i+1) for i in range(16)]},
        "普通大棚第二季": {"areas": [0.6] * 16, "names": ["E{}".format(i+1) for i in range(16)]},
        "智慧大棚第一季": {"areas": [0.6] * 4, "names": ["F{}".format(i+1) for i in range(4)]},
        "智慧大棚第二季": {"areas": [0.6] * 4, "names": ["F{}".format(i+1) for i in range(4)]}
    }

    # 获取每种地块类型允许种植的作物列表
    crop_info = {}
    for plot_type in land_info.keys():
        allowed_crops = planting_restrictions_df[planting_restrictions_df['地块类型'] == plot_type][['作物名称', '作物类型']]
        crop_info[plot_type] = allowed_crops.to_dict(orient='records')

    # 销售信息
    sales_info = {}
    for _, row in sales_results_df.iterrows():
        sales_info[row['作物名称']] = {'expected_sales': row['作物产量（销量）'], 'price': row['实际销售价格'], 'cost': row['种植成本/(元/亩)'], 'discount_price': row['实际销售价格'] * 0.5}

    # 初始化 DEAP 框架
    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
    creator.create("Individual", list, fitness=creator.FitnessMax)

    toolbox = base.Toolbox()
    toolbox.register("attr_int", lambda: randint(0, len(crop_info[list(crop_info.keys())[0]])-1) if len(crop_info[list(crop_info.keys())[0]]) > 0 else -1)
    toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_int, len(land_info.keys()) * max(len(v['areas']) for v in land_info.values()))

    toolbox.register("population", tools.initRepeat, list, toolbox.individual)

    # 适应度函数
    def evaluate(individual):
        fitness = 0
        plot_usage = {plot: 0 for plot in land_info.keys()}
```

```
        plot_crops = {plot: [] for plot in land_info.keys()}
        crop_production = {crop['作物名称']: 0 for plot in land_info.keys() for crop in crop_info[plot]}
        plots_used = {plot: False for plot in land_info.keys()}

        index = 0
        for plot, info in land_info.items():
            areas = info['areas']
            names = info['names']
            plot_type = plot
            for i, name in enumerate(names):
                crop_index = individual[index]
                index += 1
                if crop_index >= 0 and crop_index < len(crop_info[plot_type]):
                    crop = crop_info[plot_type][crop_index]
                    crop_name = crop['作物名称']
                    plot_area = areas[i % len(areas)]

                    # 检查地块的使用面积
                    if plot_usage[plot] + plot_area > sum(areas):
                        return -1,

                    plot_usage[plot] += plot_area
                    plot_crops[plot].append((crop_name, plot_area))
                    crop_production[crop_name] += plot_area

                    # 检查作物是否有销售信息
                    if crop_name not in sales_info:
                        continue

                    if crop_production[crop_name] > sales_info[crop_name]['expected_sales']:
                        regular_sales = sales_info[crop_name]['expected_sales']
                        excess_sales = crop_production[crop_name] - regular_sales
                        crop_production[crop_name] = regular_sales + excess_sales

                        # 计算利润
                        regular_profit = regular_sales * (sales_info[crop_name]['price'] - sales_info[crop_name]['cost'])
                        excess_profit = excess_sales * ((sales_info[crop_name]['price'] * 0.5) - sales_info[crop_name]['cost'])
                        fitness += regular_profit + excess_profit
                    else:
                        # 计算利润
                        profit = crop_production[crop_name] * (sales_info[crop_name]['price'] - sales_info[crop_name]['cost'])
                        fitness += profit
                    plots_used[plot] = True

```

```
        # 检查豆类作物种植
        for plot in plots_used:
            if not plots_used[plot]:
                continue

            crops_in_plot = [crop for crop, _ in plot_crops[plot]]
            if not any('豆类' in crop['作物类型'] for crop in crop_info[plot]):
                return -1,

        # 检查连续重茬种植
        for plot in plots_used:
            crops_in_plot = [crop for crop, _ in plot_crops[plot]]
            if len(set(crops_in_plot)) < len(crops_in_plot):
                return -1,

        return fitness,

    # 交叉操作
    toolbox.register("mate", tools.cxTwoPoint)
    # 变异操作
    toolbox.register("mutate", tools.mutUniformInt, low=0, up=max(len(v) for v in crop_info.values()), indpb=0.05)
    # 选择操作
    toolbox.register("select", tools.selTournament, tournsize=3)
    # 评价操作
    toolbox.register("evaluate", evaluate)

    # 主函数
    def main():
        years = range(2024, 2031)

        for year in years:
            # 初始化种群
            pop = toolbox.population(n=50)

            # 进化过程
            stats = tools.Statistics(lambda ind: ind.fitness.values)
            stats.register("avg", np.mean)
            stats.register("std", np.std)
            stats.register("min", np.min)
            stats.register("max", np.max)

            pop, logbook = algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=20, stats=stats, verbose=True)

            # 找出最佳个体
```

```
            best_ind = tools.selBest(pop, 1)[0]

            # 将结果保存到Excel
            save_results_to_excel(best_ind, year)

def save_results_to_excel(individual, year):
    results = []
    index = 0
    for plot, info in land_info.items():
        areas = info['areas']
        names = info['names']
        for i, name in enumerate(names):
            crop_index = individual[index]
            index += 1
            if crop_index >= 0 and crop_index < len(crop_info[plot]):
                crop = crop_info[plot][crop_index]
                plot_area = areas[i % len(areas)]
                results.append({
                    '年份': year,
                    '地块类型': plot,
                    '地块名称': name,
                    '种植的作物名称': crop['作物名称'],
                    '种植作物的面积': plot_area
                })

    # 创建Excel文件
    filename = f'results_{year}.xlsx'
    if os.path.exists(filename):
        mode = 'a'  # Append mode if the file exists
    else:
        mode = 'w'  # Write mode if the file does not exist

    wb = Workbook()
    ws = wb.active
    ws.append(['年份', '地块类型', '地块名称', '种植的作物名称', '种植作物的面积'])

    for result in results:
        ws.append([
            result['年份'],
            result['地块类型'],
            result['地块名称'],
            result['种植的作物名称'],
            result['种植作物的面积']
        ])
```

```
    try:
        wb.save(filename)
    except PermissionError:
        raise PermissionError(f"Could not write to '{filename}'. Please close any open instances of this file.")

if __name__ == "__main__":
    main()
```

9.3 问题 2

（一）单季作物

```
import pandas as pd
import numpy as np
import random
from deap import base, creator, tools, algorithms

# 读取 Excel 文件
sales_results_path = r'D:\数模\C 题\第二问草稿\单季\使用数据\连接结果.xlsx'
crop_types_path = r'D:\数模\C 题\第二问草稿\单季\使用数据\每种作物适合种植的地块类型与季别_合并.xlsx'
crop_varieties_path = r'D:\数模\C 题\第二问草稿\单季\使用数据\作物种类.xlsx'

sales_results_df = pd.read_excel(sales_results_path, sheet_name='2023 销售结果')
analysis_df = pd.read_excel(sales_results_path, sheet_name='销售结果分析')
crop_types_df = pd.read_excel(crop_types_path)
crop_varieties_df = pd.read_excel(crop_varieties_path, sheet_name='第一季（单）')

# 读取 Excel 文件中的数据
file_path = r"D:\数模\C 题\第二问草稿\单季\使用数据\作物种类.xlsx"
sheet_name = "第一季（单）"

crop_varieties_df = pd.read_excel(file_path, sheet_name=sheet_name)

# 生成 valid_crops 字典，存储有效的作物及其是否为豆类
valid_crops = {}  # 存储有效的作物及其是否为豆类
for index, row in crop_varieties_df.iterrows():
    crop_name = row['作物名称']
    crop_type = row['作物类型']
    is_legume = '豆类' in crop_type
    valid_crops[crop_name] = is_legume

# 2023 年的实际种植数据
actual_planting_2023 = {}

# 定义需要保留的字段列表
fields_to_keep = ['作物名称', '作物类型', '地块名称', '亩产量/斤', '地块类型', '种植成本/(元/斤)',
```

```
                    '种植面积/亩', '作物产量（销量）', '实际销售价格']

    # 遍历2023年的销售结果数据
    for index, row in sales_results_df.iterrows():
        plot_name = row['地块名称']
        crop_name = row['作物名称']

        # 查找作物类型和其他相关信息
        crop_info = sales_results_df[sales_results_df['作物名称'] == crop_name]

        if not crop_info.empty:
            # 获取作物类型
            crop_type = crop_info.iloc[0]['作物类型']

            # 检查是否为豆类
            is_legume = '豆类' in crop_type

            # 获取其他相关信息
            for field in fields_to_keep:
                value = row[field] if field != '作物类型' else crop_type
                setattr(row, f'field_{field}', value)  # 使用setattr动态设置属性以便后续访问

            # 存储到字典中
            actual_planting_2023[(plot_name, crop_name)] = {f'field_{field}': getattr(row, f'field_{field}') for field in fields_to_keep}


    # 提取地块面积
    land_areas = {
        '平旱地': {'A1': 80, 'A2': 55, 'A3': 35, 'A4': 72, 'A5': 68, 'A6': 55},
                                                                    '梯田': {f'B{i+1}': 60 if i == 0 else 46 if i == 1 else 40 if i == 2 else 28 if i == 3 else 25 if i == 4 else 86 if i == 5 else 55 if i == 6 else 44 if i == 7 else 50 if i == 8 else 25 if i == 9 else 60 if i == 10 else 45 if i == 11 else 35 if i == 12 else 20 for i in range(14)},
        '山坡地': {'C1': 15, 'C2': 13, 'C3': 15, 'C4': 18, 'C5': 27, 'C6': 20}
    }

    # 不确定性信息
    sales_growth_rate = {
        '小麦': (0.05, 0.10),
        '玉米': (0.05, 0.10),
        '其他': 0.05,
    }
    yield_variation = 0.10
    cost_growth_rate = 0.05
    price_growth_rate = {
```

```
        '粮食类': 0.00,
        '蔬菜类': 0.05,
        '食用菌': (-0.05, -0.01),
        '羊肚菌': -0.05,
    }

    # 蒙特卡洛模拟
    def monte_carlo_simulation(actual_data, years):
        simulated_data = []
        for year in range(years):
            temp_data = {}
            for (plot, crop), data in actual_data.items():
                # 销售量
                if crop in ['小麦', '玉米']:
                    sale = data['field_作物产量（销量）'] * (1 + np.random.uniform(sales_growth_rate[crop][0], sales_growth_rate[crop][1]) * year)
                else:
                    sale = data['field_作物产量（销量）'] * (1 + np.random.uniform(-sales_growth_rate['其他'], sales_growth_rate['其他']))

                # 亩产量
                yield_per_acre = data['field_亩产量/斤'] * (1 + np.random.uniform(-yield_variation, yield_variation))

                # 种植成本
                cost_per_acre = data['field_种植成本/(元/斤)'] * (1 + cost_growth_rate * year)

                # 销售价格
                if data['field_作物类型'] == '粮食类':
                    selling_price = data['field_实际销售价格'] * (1 + price_growth_rate['粮食类'] * year)
                elif data['field_作物类型'] == '蔬菜类':
                    selling_price = data['field_实际销售价格'] * (1 + price_growth_rate['蔬菜类'] * year)
                elif data['field_作物类型'] == '食用菌':
                    if crop == '羊肚菌':
                        selling_price = data['field_实际销售价格'] * (1 + price_growth_rate['羊肚菌'] * year)
                    else:
                        selling_price = data['field_实际销售价格'] * (1 + np.random.uniform(price_growth_rate['食用菌'][0], price_growth_rate['食用菌'][1]))
                else:
                    # 处理未分类的作物类型
                    selling_price = data['field_实际销售价格'] * (1 + np.random.uniform(-0.05, 0.05) * year)

                temp_data[crop] = {
                    'yield_per_acre': yield_per_acre,
                    'selling_price': selling_price,
                    'cost_per_acre': cost_per_acre,
```

```
                }
            simulated_data.append(temp_data)
        return simulated_data

    # 提取作物相关数据
    simulated_data = monte_carlo_simulation(actual_planting_2023, 7)

    # 生成多种情景
    def generate_scenarios(simulated_data, years):
        scenarios = []
        for year in range(len(years)):
            temp_data = {}
            for crop, values in simulated_data[year].items():
                # 预期销售量
                if crop in ['小麦', '玉米']:
                    temp_data[crop] = {
                        'yield_per_acre': values['yield_per_acre'] * (1 + np.random.uniform(sales_growth_rate[crop][0], sales_growth_rate[cro
    p][1])),
                        'selling_price': values['selling_price'] * (1 + np.random.uniform(price_growth_rate['粮食类'], price_growth_rate['粮食
    类'])),
                        'cost_per_acre': values['cost_per_acre'] * (1 + np.random.uniform(cost_growth_rate, cost_growth_rate)),
                    }
                elif crop == '羊肚菌':
                    temp_data[crop] = {
                        'yield_per_acre': values['yield_per_acre'] * (1 + np.random.uniform(-yield_variation, yield_variation)),
                        'selling_price': values['selling_price'] * (1 - np.random.uniform(price_growth_rate['羊肚菌'], price_growth_rate['羊肚
    菌'])),
                        'cost_per_acre': values['cost_per_acre'] * (1 + np.random.uniform(cost_growth_rate, cost_growth_rate)),
                    }
                else:
                    temp_data[crop] = {
                        'yield_per_acre': values['yield_per_acre'] * (1 + np.random.uniform(-yield_variation, yield_variation)),
                        'selling_price': values['selling_price'] * (1 + np.random.uniform(price_growth_rate['食用菌'][0], price_growth_rate['
    食用菌'][1])),
                        'cost_per_acre': values['cost_per_acre'] * (1 + np.random.uniform(cost_growth_rate, cost_growth_rate)),
                    }
            scenarios.append(temp_data)
        return scenarios

    # 生成情景
    scenarios = generate_scenarios(simulated_data, range(2024, 2031))

    # 初始化 DEAP 环境
    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
```

```
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()

# 计算个体长度
num_plots = sum([len(plots) for plots in land_areas.values()])
num_years = len(scenarios)
num_crops = len(valid_crops)
individual_length = num_plots * num_years * num_crops

toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=individual_length)

# 检查约束条件
def check_constraints(decoded_individual, land_areas, valid_crops, actual_planting_2023):
    # 耕地面积限制
    for year in range(len(scenarios)):
        # 创建一个字典来存储每个地块的总种植面积
        plot_total_areas = {}

        for land_type, plots in land_areas.items():
            for plot, area in plots.items():
                # 计算每个地块的总种植面积
                if plot not in plot_total_areas:
                    plot_total_areas[plot] = 0

                for crop in valid_crops:
                    plot_total_areas[plot] += decoded_individual.get((land_type, plot, crop, year), 0)

                # 检查是否超出面积限制
                if plot_total_areas[plot] > area:
                    return False

    # 不重茬约束
    crops = [None] * len(scenarios)
    for year in range(len(scenarios)):
        for crop in valid_crops:
          if decoded_individual[(land_type, plot, crop, year)]:
              crops[year] = crop
              break
        for year in range(len(scenarios) - 1):
              if crops[year] == crops[year + 1]:
                  return False

```

```
                # 豆类作物三年内种植一次的约束
        legume_years = [year for year in range(len(scenarios)) if any(valid_crops[crop] for crop in valid_crops if decoded_individual[(land_type, plot, crop, year)])]
        if len(legume_years) > 0 and max(legume_years) - min(legume_years) > 2:
            return False

        # 每种作物每季的种植地不能太分散
        for crop in valid_crops:
            for year in range(len(scenarios)):
                plots_per_type = {}
                for land_type, plots in land_areas.items():
                    planted_plots = [plot for plot in plots if decoded_individual[(land_type, plot, crop, year)]]
                    if planted_plots:
                        plots_per_type[land_type] = planted_plots

                for land_type, planted_plots in plots_per_type.items():
                    if len(planted_plots) > 1:
                        if not is_consecutive(planted_plots):
                            return False
        return True


    # 适应度函数
    def evaluate(individual, scenarios, land_areas, valid_crops, actual_planting_2023):
        total_profit = 0
        decoded_individual = decode_individual(individual, num_years, land_areas, valid_crops)

        # 检查约束条件
        if not check_constraints(decoded_individual, land_areas, valid_crops, actual_planting_2023):
            return -1,

        # 计算总利润
        for year in range(len(scenarios)):
            for land_type, plots in land_areas.items():
                for plot, area in plots.items():
                    for crop, data in scenarios[year].items():
                        if decoded_individual[(land_type, plot, crop, year)] > 0:
                            profit = (data['yield_per_acre'] * data['selling_price'] - data['cost_per_acre']*data['yield_per_acre']) * decoded_individual[(land_type, plot, crop, year)]
                            total_profit += profit
        return total_profit,

    # 解码个体
    def decode_individual(individual, num_years, land_areas, valid_crops):
```

```
        decoded_individual = {}
        index = 0
        for year in range(num_years):
            for land_type, plots in land_areas.items():
                for plot in plots:
                    for crop in valid_crops:
                        # 假设每个基因代表作物在地块上的种植比例
                        decoded_individual[(land_type, plot, crop, year)] = individual[index] / len(valid_crops) * land_areas[land_type][plot]
                        index += 1
        return decoded_individual

    # 判断地块名称是否连续
    def is_consecutive(plots):
        plots_sorted = sorted([int(plot[1:]) for plot in plots])
        return all(plots_sorted[i] + 1 == plots_sorted[i + 1] for i in range(len(plots_sorted) - 1))


    # 交叉操作
    toolbox.register("mate", tools.cxTwoPoint)
    # 变异操作
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
    # 选择操作
    toolbox.register("select", tools.selTournament, tournsize=3)
    # 注册评估函数
    toolbox.register("evaluate", evaluate, scenarios=scenarios, land_areas=land_areas, valid_crops=valid_crops, actual_planting_2023=actual_planting_2023)

    # 初始化种群
    POPULATION_SIZE = 100
    HALL_OF_FAME_SIZE = 1
    population = [toolbox.individual() for _ in range(POPULATION_SIZE)]
    hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

    # 进化过程
    NGEN = 250
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", np.mean)
    stats.register("std", np.std)
    stats.register("min", np.min)
    stats.register("max", np.max)

    population, logbook = algorithms.eaSimple(population, toolbox, cxpb=0.6, mutpb=0.005, ngen=NGEN, stats=stats, halloffame=hof, verbose=True)
```

```

# 输出最优解
best_individual = hof[0]
decoded_best_individual = decode_individual(best_individual, num_years, land_areas, valid_crops)

print("Best Individual Fitness:", best_individual.fitness.values)
print("Decoded Best Individual:")

# 提取最优解中的具体种植面积信息
results = []
for year in range(len(scenarios)):
    for land_type, plots in land_areas.items():
        for plot in plots:
            for crop in valid_crops:
                # 获取特定年份、土地类型、地块和作物的种植面积
                planting_area = decoded_best_individual.get((land_type, plot, crop, year), 0)
                if planting_area > 0:
                    results.append({
                        '年份': 2024 + year,
                        '地块类型': land_type,
                        '地块名称': plot,
                        '作物名称': crop,
                        '种植面积': planting_area
                    })

# 保存结果到Excel
output_path = r'D:\数模\C 题\第二问草稿\单季\种植方案.xlsx'
output_df = pd.DataFrame(results)
with pd.ExcelWriter(output_path) as writer:
    output_df.to_excel(writer, sheet_name='种植方案', index=False)

print(f"Results saved to {output_path}")
```

（二）双季作物

```
import pandas as pd
import numpy as np
import random
from deap import base, creator, tools, algorithms

# 读取 Excel 文件
sales_results_path = r'D:\数模\C 题\第二问草稿\双季\使用数据\连接结果（双季）.xlsx'
crop_types_path = r'D:\数模\C 题\第二问草稿\双季\使用数据\每种作物适合种植的地块类型与季别_合并.xlsx'
crop_varieties_path = r'D:\数模\C 题\第二问草稿\双季\使用数据\作物种植限制.xlsx'

sales_results_df = pd.read_excel(sales_results_path, sheet_name='2023 销售结果')
```

```
analysis_df = pd.read_excel(sales_results_path, sheet_name='销售结果分析')
crop_types_df = pd.read_excel(crop_types_path)
crop_varieties_df = pd.read_excel(crop_varieties_path)

# 生成 valid_crops 字典，存储有效的作物及其是否为豆类
valid_crops_s1 = {}
valid_crops_s2 = {}
valid_crops_p1 = {}
valid_crops_p2 = {}
valid_crops_z1 = {}
valid_crops_z2 = {}

for index, row in crop_varieties_df.iterrows():
    if row['地块类型'] == '水浇地第一季':
        crop_name = row['作物名称']
        crop_type = row['作物类型']
        is_legume = '豆类' in crop_type
        valid_crops_s1[crop_name] = is_legume
    elif row['地块类型'] == '水浇地第二季':
        crop_name = row['作物名称']
        crop_type = row['作物类型']
        is_legume = '豆类' in crop_type
        valid_crops_s2[crop_name] = is_legume

for index, row in crop_varieties_df.iterrows():
    if row['地块类型'] == '普通大棚第一季':
        crop_name = row['作物名称']
        crop_type = row['作物类型']
        is_legume = '豆类' in crop_type
        valid_crops_p1[crop_name] = is_legume
    elif row['地块类型'] == '普通大棚第二季':
        crop_name = row['作物名称']
        crop_type = row['作物类型']
        is_legume = '豆类' in crop_type
        valid_crops_p2[crop_name] = is_legume

for index, row in crop_varieties_df.iterrows():
    if row['地块类型'] == '智慧大棚第一季':
        crop_name = row['作物名称']
        crop_type = row['作物类型']
        is_legume = '豆类' in crop_type
        valid_crops_z1[crop_name] = is_legume
    elif row['地块类型'] == '智慧大棚第二季':
        crop_name = row['作物名称']
```

```
            crop_type = row['作物类型']
            is_legume = '豆类' in crop_type
            valid_crops_z2[crop_name] = is_legume

    # 2023 年的实际种植数据
    actual_planting_2023 = {}
    fields_to_keep = ['作物名称', '作物类型', '地块名称', '亩产量/斤', '地块类型', '种植成本/(元/斤)', '种植面积/亩', '作物产量（销量）', '实际销售价格']

    for index, row in sales_results_df.iterrows():
        plot_name = row['地块名称']
        crop_name = row['作物名称']
        crop_info = sales_results_df[sales_results_df['作物名称'] == crop_name]
        if not crop_info.empty:
            crop_type = crop_info.iloc[0]['作物类型']
            is_legume = '豆类' in crop_type
            for field in fields_to_keep:
                value = row[field] if field != '作物类型' else crop_type
                setattr(row, f'field_{field}', value)
            actual_planting_2023[(plot_name, crop_name)] = {f'field_{field}': getattr(row, f'field_{field}') for field in fields_to_keep}

    # 地块面积信息
    land_areas = {
        '水浇地第一季': {'D1': 15, 'D2': 10, 'D3': 14, 'D4': 6, 'D5': 10, 'D6': 12, 'D7': 22, 'D8': 20},
        '普通大棚第一季': {'E1': 0.6, 'E2': 0.6, 'E3': 0.6, 'E4': 0.6, 'E5': 0.6, 'E6': 0.6, 'E7': 0.6, 'E8': 0.6,
                      'E9': 0.6, 'E10': 0.6, 'E11': 0.6, 'E12': 0.6, 'E13': 0.6, 'E14': 0.6, 'E15': 0.6, 'E16': 0.6},
        '智慧大棚第一季': {'F1': 0.6, 'F2': 0.6, 'F3': 0.6, 'F4': 0.6},
        '水浇地第二季': {'D1': 15, 'D2': 10, 'D3': 14, 'D4': 6, 'D5': 10, 'D6': 12, 'D7': 22, 'D8': 20},
        '普通大棚第二季': {'E1': 0.6, 'E2': 0.6, 'E3': 0.6, 'E4': 0.6, 'E5': 0.6, 'E6': 0.6, 'E7': 0.6, 'E8': 0.6,
                      'E9': 0.6, 'E10': 0.6, 'E11': 0.6, 'E12': 0.6, 'E13': 0.6, 'E14': 0.6, 'E15': 0.6, 'E16': 0.6},
        '智慧大棚第二季': {'F1': 0.6, 'F2': 0.6, 'F3': 0.6, 'F4': 0.6}
    }

    # 创建 DEAP 的 creator 对象
    try:
        del creator.FitnessMax
    except AttributeError:
        pass

    try:
        del creator.Individual
    except AttributeError:
        pass
```

```
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

# 创建 toolbox
toolbox = base.Toolbox()

# 注册生成器
toolbox.register("attr_bool", random.randint, 0, 1)

# 计算个体长度
num_plots = sum([len(plots) for plots in land_areas.values()])
num_years = 7  # 2024-2030
individual_length = num_plots * num_years * len(valid_crops_s1)

toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=individual_length)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# 蒙特卡洛模拟
def monte_carlo_simulation(actual_data, years):
    simulated_data = []
    for year in range(years):
        temp_data = {}
        for (plot, crop), data in actual_data.items():
            # 销售量
            if crop in ['小麦', '玉米']:
                sale = data['field_作物产量（销量）'] * (1 + np.random.uniform(0.05, 0.10) * year)
            else:
                sale = data['field_作物产量（销量）'] * (1 + np.random.uniform(-0.05, 0.05))

            # 亩产量
            yield_per_acre = data['field_亩产量/斤'] * (1 + np.random.uniform(-0.10, 0.10))

            # 种植成本
            cost_per_acre = data['field_种植成本/(元/斤)'] * (1 + 0.05 * year)

            # 销售价格
            if data['field_作物类型'] == '粮食类':
                selling_price = data['field_实际销售价格'] * (1 + 0.01 * year)
            elif data['field_作物类型'] == '蔬菜类':
                selling_price = data['field_实际销售价格'] * (1 + 0.05 * year)
            elif data['field_作物类型'] == '食用菌':
                if crop == '羊肚菌':
                    selling_price = data['field_实际销售价格'] * (1 - 0.05 * year)
                else:
```

```
                    selling_price = data['field_实际销售价格'] * (1 + np.random.uniform(-0.01, -0.05))
            else:
                # 处理未分类的作物类型
                selling_price = data['field_实际销售价格'] * (1 + np.random.uniform(-0.05, 0.05) * year)

            temp_data[(plot, crop)] = {
                'yield_per_acre': yield_per_acre,
                'selling_price': selling_price,
                'cost_per_acre': cost_per_acre,
                'sale': sale
            }
        simulated_data.append(temp_data)
    return simulated_data

# 生成多种情景
def generate_scenarios(simulated_data, years):
    scenarios = []
    for year in years:
        temp_data = {}
        for (plot, crop), values in simulated_data[year - 2024].items():
            # 预期销售量
            if crop in ['小麦', '玉米']:
                temp_data[(plot, crop)] = {
                    'yield_per_acre': values['yield_per_acre'] * (1 + np.random.uniform(0.05, 0.10)),
                    'selling_price': values['selling_price'] * (1 + np.random.uniform(0.01, 0.01)),
                    'cost_per_acre': values['cost_per_acre'] * (1 + np.random.uniform(0.05, 0.05)),
                    'sale': values['sale']
                }
            elif crop == '羊肚菌':
                temp_data[(plot, crop)] = {
                    'yield_per_acre': values['yield_per_acre'] * (1 + np.random.uniform(-0.10, 0.10)),
                    'selling_price': values['selling_price'] * (1 - np.random.uniform(0.05, 0.05)),
                    'cost_per_acre': values['cost_per_acre'] * (1 + np.random.uniform(0.05, 0.05)),
                    'sale': values['sale']
                }
            else:
                temp_data[(plot, crop)] = {
                    'yield_per_acre': values['yield_per_acre'] * (1 + np.random.uniform(-0.10, 0.10)),
                    'selling_price': values['selling_price'] * (1 + np.random.uniform(-0.01, -0.05)),
                    'cost_per_acre': values['cost_per_acre'] * (1 + np.random.uniform(0.05, 0.05)),
                    'sale': values['sale']
                }
        scenarios.append(temp_data)
    return scenarios
```

```
187.
188.     # 使用蒙特卡洛模拟生成未来7年的数据
189.     simulated_data = monte_carlo_simulation(actual_planting_2023, 7)
190.
191.     # 生成多种情景
192.     years = range(2024, 2031)
193.     scenarios = generate_scenarios(simulated_data, years)
194.
195.     # 适应度函数
196.     def evaluate(individual, scenarios, land_areas, valid_crops_s1, valid_crops_s2, valid_crops_p1, valid_crops_p2, valid_crops_z1, valid_cro
    ps_z2, actual_planting_2023):
197.         total_profit = 0
198.         decoded_individual = decode_individual(individual, num_plots, num_years, land_areas, valid_crops_s1, valid_crops_s2, valid_crops_p1,
    valid_crops_p2, valid_crops_z1, valid_crops_z2)
199.
200.         # 检查约束条件
201.         if not check_constraints(decoded_individual, land_areas, valid_crops_s1, valid_crops_s2, valid_crops_p1, valid_crops_p2, valid_crops_
    z1, valid_crops_z2, scenarios):
202.             return -1,
203.
204.         # 计算总利润
205.         for year in range(len(scenarios)):
206.             for land_type, plots in land_areas.items():
207.                 valid_crops_for_type = get_valid_crops(land_type, year)
208.                 for plot, area in plots.items():
209.                     for crop in valid_crops_for_type:
210.                         if decoded_individual.get((land_type, plot, crop, year), 0) > 0:
211.                             profit = (scenarios[year][(plot, crop)]['yield_per_acre'] * scenarios[year][(plot, crop)]['selling_price'] - scen
    arios[year][(plot, crop)]['cost_per_acre']*scenarios[year][(plot, crop)]['yield_per_acre']) * decoded_individual.get((land_type, plot, crop, yea
    r), 0)
212.                             total_profit += profit
213.         return total_profit,
214.
215.     def decode_individual(individual, num_plots, num_years, land_areas, valid_crops_s1, valid_crops_s2, valid_crops_p1, valid_crops_p2, valid
    _crops_z1, valid_crops_z2):
216.         decoded_individual = {}
217.         plot_counter = 0  # 用于追踪地块的计数器
218.
219.         # 遍历所有地块
220.         for land_type, plots in land_areas.items():
221.             for plot, area in plots.items():
222.                 for year in range(num_years):
223.                     valid_crops = get_valid_crops(land_type, year)
224.
```

```
                # 对于每一种有效作物，检查个体中是否有种植标记
                for j, crop in enumerate(valid_crops):
                    key = (land_type, plot, crop, year)
                    if key not in decoded_individual:
                        decoded_individual[key] = 0

                    # 通过索引找到对应的基因位
                    gene_index = plot_counter * len(valid_crops) + j
                    if individual[gene_index]:
                        decoded_individual[key] = 1

                # 更新地块计数器
                plot_counter += 1

            # 每个地块类型中的地块都处理完后，重置计数器
            if plot_counter >= num_plots:
                plot_counter = 0

    return decoded_individual

# 获取对应地块类型的作物
def get_valid_crops(land_type, year):
    if '水浇地第一季' in land_type:
        return valid_crops_s1
    elif '水浇地第二季' in land_type:
        return valid_crops_s2
    elif '普通大棚第一季' in land_type:
        return valid_crops_p1
    elif '普通大棚第二季' in land_type:
        return valid_crops_p2
    elif '智慧大棚第一季' in land_type:
        return valid_crops_z1
    elif '智慧大棚第二季' in land_type:
        return valid_crops_z2
    return {}

# 检查约束条件
def check_constraints(decoded_individual, land_areas, valid_crops_s1, valid_crops_s2, valid_crops_p1, valid_crops_p2, valid_crops_z1, val
id_crops_z2, scenarios):
    # 耕地面积限制
    for year in range(len(scenarios)):
        plot_total_areas = {}
        for land_type, plots in land_areas.items():
            valid_crops_for_type = get_valid_crops(land_type, year)
```

```
                for plot, area in plots.items():
                    plot_total_areas.setdefault(plot, 0)
                    for crop in valid_crops_for_type:
                        plot_total_areas[plot] += decoded_individual.get((land_type, plot, crop, year), 0)
                    if plot_total_areas[plot] > area:
                        return False

        # 不重茬约束
        for land_type, plots in land_areas.items():
            for plot in plots:
                crops = [None] * len(scenarios)
                for year in range(len(scenarios)):
                    valid_crops_for_type = get_valid_crops(land_type, year)
                    for crop in valid_crops_for_type:
                        if decoded_individual.get((land_type, plot, crop, year), 0) > 0:
                            crops[year] = crop
                            break
                for year in range(len(scenarios) - 1):
                    if crops[year] == crops[year + 1]:
                        return False

        # 豆类作物三年内种植一次的约束
        for land_type, plots in land_areas.items():
            valid_crops_for_type = get_valid_crops(land_type, year)
            for plot in plots:
                legume_years = [year for year in range(len(scenarios)) if any(valid_crops_for_type[crop] for crop in valid_crops_for_type if decoded_individual.get((land_type, plot, crop, year), 0) > 0)]
                if legume_years and max(legume_years) - min(legume_years) > 3:
                    return False

        # 每种作物每季的种植地不能太分散
        for crop in valid_crops_s1.keys():  # 假设所有作物都在第一季种植
            for year in range(len(scenarios)):
                plots_per_type = {}
                for land_type, plots in land_areas.items():
                    valid_crops_for_type = get_valid_crops(land_type, year)
                    if crop in valid_crops_for_type:
                        planted_plots = [plot for plot in plots if decoded_individual.get((land_type, plot, crop, year), 0) > 0]
                        if planted_plots:
                            plots_per_type[land_type] = planted_plots
                for land_type, planted_plots in plots_per_type.items():
                    if len(planted_plots) > 1:
                        if not is_consecutive(planted_plots):
                            return False
```

```
            return True

    # 判断地块名称是否连续
    def is_consecutive(plots):
        plots_sorted = sorted([int(plot[1:]) for plot in plots])  # 假设地块名为 D1, D2, D3 ...
        return all(plots_sorted[i] + 1 == plots_sorted[i + 1] for i in range(len(plots_sorted) - 1))

    # 注册 evaluate 函数
    toolbox.register("evaluate", evaluate, scenarios=scenarios, land_areas=land_areas, valid_crops_s1=valid_crops_s1, valid_crops_s2=valid_crops_s2, valid_crops_p1=valid_crops_p1, valid_crops_p2=valid_crops_p2, valid_crops_z1=valid_crops_z1, valid_crops_z2=valid_crops_z2, actual_planting_2023=actual_planting_2023)

    # 注册其他操作
    toolbox.register("mate", tools.cxTwoPoint)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
    toolbox.register("select", tools.selTournament, tournsize=3)

    # 初始化种群
    POPULATION_SIZE = 100
    HALL_OF_FAME_SIZE = 1
    population = toolbox.population(n=POPULATION_SIZE)
    hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

    # 进化过程
    NGEN = 250
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", np.mean)
    stats.register("std", np.std)
    stats.register("min", np.min)
    stats.register("max", np.max)

    population, logbook = algorithms.eaSimple(population, toolbox, cxpb=0.6, mutpb=0.005, ngen=NGEN, stats=stats, halloffame=hof, verbose=True)

    # 输出最优解
    best_individual = hof[0]
    decoded_best_individual = decode_individual(best_individual, num_plots, num_years, land_areas, valid_crops_s1, valid_crops_s2, valid_crops_p1, valid_crops_p2, valid_crops_z1, valid_crops_z2)

    print("Best Individual Fitness:", best_individual.fitness.values)
    print("Decoded Best Individual:")

    # 提取最优解中的具体种植面积信息
    results = []
```

```
for year in range(len(scenarios)):
    for land_type, plots in land_areas.items():
        for plot in plots:
            for crop in get_valid_crops(land_type, year):
                # 获取特定年份、土地类型、地块和作物的种植面积
                planting_area = decoded_best_individual.get((land_type, plot, crop, year), 0)
                if planting_area > 0:
                    results.append({
                        '年份': 2024 + year,
                        '地块类型': land_type,
                        '地块名称': plot,
                        '作物名称': crop,
                        '种植面积': planting_area
                    })

# 保存结果到Excel
output_path = r'D:\数模\C 题\第二问草稿\双季\种植方案.xlsx'
output_df = pd.DataFrame(results)
with pd.ExcelWriter(output_path) as writer:
    output_df.to_excel(writer, sheet_name='种植方案', index=False)

print(f"Results saved to {output_path}")
```

9.4 问题 3

```
import pandas as pd
import numpy as np
import random
from deap import base, creator, tools, algorithms
import functools

# 读取 Excel 文件
sales_results_path = r'C:\Users\HP\Desktop\连接结果(2).xlsx'
crop_types_path = r'C:\Users\HP\Desktop\每种作物适合种植的地块类型与季别_合并.xlsx'
crop_varieties_path = r'C:\Users\HP\Desktop\作物种类.xlsx'

# 读取替代性与互补性数据
substitute_complement_path = r'C:\Users\HP\Desktop\单季作物的替代性与互补性.xlsx'
substitute_complement_df = pd.read_excel(substitute_complement_path)

# 读取相关性数据
correlation_path = r'C:\Users\HP\Desktop\单季作物的产量价格成本相关性.xlsx'
correlation_df = pd.read_excel(correlation_path)

# 处理缺失数据
# 例如，将缺失的相关性值替换为0或其他作物的平均值
```

```
correlation_df.fillna(0, inplace=True)  # 这里假设缺失值用0代替，你可以根据实际情况调整


sales_results_df = pd.read_excel(sales_results_path, sheet_name='2023销售结果')
analysis_df = pd.read_excel(sales_results_path, sheet_name='销售结果分析')
crop_types_df = pd.read_excel(crop_types_path)
crop_varieties_df = pd.read_excel(crop_varieties_path, sheet_name='第一季（单）')

# 读取 Excel 文件中的数据
file_path = r"C:\Users\HP\Desktop\作物种类.xlsx"
sheet_name = "第一季（单）"

crop_varieties_df = pd.read_excel(file_path, sheet_name=sheet_name)


# 生成 valid_crops 字典，存储有效的作物及其是否为豆类
valid_crops = {}  # 存储有效的作物及其是否为豆类
for index, row in crop_varieties_df.iterrows():
    crop_name = row['作物名称']
    crop_type = row['作物类型']
    is_legume = '豆类' in crop_type
    valid_crops[crop_name] = is_legume


# 将替代性和互补性关系转换为字典
substitute_complement = {}
for index, row in substitute_complement_df.iterrows():
    crop1 = row['作物1']
    crop2 = row['作物2']
    relation = row['相关系数']
    substitute_complement[(crop1, crop2)] = relation

# 将相关性数据转换为字典
correlation_data = {}
for index, row in correlation_df.iterrows():
    crop_name = row['作物名称']
    correlation_data[crop_name] = {
        '产量-销售价格': row['产量-销售价格'],
        '产量-种植成本': row['产量-种植成本'],
        '价格-种植成本': row['价格-种植成本']
    }


# 2023年的实际种植数据
```

```
actual_planting_2023 = {}

# 定义需要保留的字段列表
fields_to_keep = ['作物名称', '作物类型', '地块名称', '亩产量/斤', '地块类型', '种植成本/(元/亩)',
                  '种植面积/亩', '作物产量（销量）', '实际销售价格']

# 遍历2023年的销售结果数据
for index, row in sales_results_df.iterrows():
    plot_name = row['地块名称']
    crop_name = row['作物名称']

    # 查找作物类型和其他相关信息
    crop_info = sales_results_df[sales_results_df['作物名称'] == crop_name]

    if not crop_info.empty:
        # 获取作物类型
        crop_type = crop_info.iloc[0]['作物类型']

        # 检查是否为豆类
        is_legume = '豆类' in crop_type

        # 获取其他相关信息
        for field in fields_to_keep:
            value = row[field] if field != '作物类型' else crop_type
            setattr(row, f'field_{field}', value)  # 使用setattr动态设置属性以便后续访问

        # 存储到字典中
        actual_planting_2023[(plot_name, crop_name)] = {f'field_{field}': getattr(row, f'field_{field}') for field in
                                                        fields_to_keep}

# 提取地块面积
land_areas = {
    '平旱地': {'A1': 80, 'A2': 55, 'A3': 35, 'A4': 72, 'A5': 68, 'A6': 55},
    '梯田': {
        f'B{i + 1}': 60 if i == 0 else 46 if i == 1 else 40 if i == 2 else 28 if i == 3 else 25 if i == 4 else 86 if i == 5 else 55 if i == 6 else 44 if i == 7 else 50 if i == 8 else 25 if i == 9 else 60 if i == 10 else 45 if i == 11 else 35 if i == 12 else 20
        for i in range(14)},
    '山坡地': {'C1': 15, 'C2': 13, 'C3': 15, 'C4': 18, 'C5': 27, 'C6': 20}
}

# 不确定性信息
sales_growth_rate = {
    '小麦': (0.05, 0.10),
    '玉米': (0.05, 0.10),
```

```
    '其他': 0.05,
}
yield_variation = 0.10
cost_growth_rate = 0.05
price_growth_rate = {
    '粮食类': 0.00,
    '蔬菜类': 0.05,
    '食用菌': (-0.05, -0.01),
    '羊肚菌': -0.05,
}


# 蒙特卡洛模拟
def monte_carlo_simulation(actual_data, years):
    simulated_data = []
    for year in range(years):
        temp_data = {}
        for (plot, crop), data in actual_data.items():
            # 销售量
            if crop in ['小麦', '玉米']:
                sale = data['field_作物产量（销量）'] * (
                            1 + np.random.uniform(sales_growth_rate[crop][0], sales_growth_rate[crop][1]) * year)
            else:
                sale = data['field_作物产量（销量）'] * (
                            1 + np.random.uniform(-sales_growth_rate['其他'], sales_growth_rate['其他']))

            # 亩产量
            yield_per_acre = data['field_亩产量/斤'] * (1 + np.random.uniform(-yield_variation, yield_variation))

            # 种植成本
            cost_per_acre = data['field_种植成本/(元/亩)'] * (1 + cost_growth_rate * year)

            # 销售价格
            if data['field_作物类型'] == '粮食类':
                selling_price = data['field_实际销售价格'] * (1 + price_growth_rate['粮食类'] * year)
            elif data['field_作物类型'] == '蔬菜类':
                selling_price = data['field_实际销售价格'] * (1 + price_growth_rate['蔬菜类'] * year)
            elif data['field_作物类型'] == '食用菌':
                if crop == '羊肚菌':
                    selling_price = data['field_实际销售价格'] * (1 + price_growth_rate['羊肚菌'] * year)
                else:
                    selling_price = data['field_实际销售价格'] * (
                                1 + np.random.uniform(price_growth_rate['食用菌'][0], price_growth_rate['食用菌'][1]))
            else:
```

```
                # 处理未分类的作物类型
                selling_price = data['field_实际销售价格'] * (1 + np.random.uniform(-0.05, 0.05) * year)

            temp_data[crop] = {
                'yield_per_acre': yield_per_acre,
                'selling_price': selling_price,
                'cost_per_acre': cost_per_acre,
            }
        simulated_data.append(temp_data)
    return simulated_data


# 提取作物相关数据
simulated_data = monte_carlo_simulation(actual_planting_2023, 7)


# 生成多种情景
def generate_scenarios(simulated_data, years):
    scenarios = []
    for year in range(len(years)):
        temp_data = {}
        for crop, values in simulated_data[year].items():
            # 预期销售量
            if crop in ['小麦', '玉米']:
                temp_data[crop] = {
                    'yield_per_acre': values['yield_per_acre'] * (
                                1 + np.random.uniform(sales_growth_rate[crop][0], sales_growth_rate[crop][1])),
                    'selling_price': values['selling_price'] * (
                                1 + np.random.uniform(price_growth_rate['粮食类'], price_growth_rate['粮食类'])),
                    'cost_per_acre': values['cost_per_acre'] * (
                                1 + np.random.uniform(cost_growth_rate, cost_growth_rate)),
                }
            elif crop == '羊肚菌':
                temp_data[crop] = {
                    'yield_per_acre': values['yield_per_acre'] * (
                                1 + np.random.uniform(-yield_variation, yield_variation)),
                    'selling_price': values['selling_price'] * (
                                1 - np.random.uniform(price_growth_rate['羊肚菌'], price_growth_rate['羊肚菌'])),
                    'cost_per_acre': values['cost_per_acre'] * (
                                1 + np.random.uniform(cost_growth_rate, cost_growth_rate)),
                }
            else:
                temp_data[crop] = {
                    'yield_per_acre': values['yield_per_acre'] * (
```

```
                                1 + np.random.uniform(-yield_variation, yield_variation)),
                    'selling_price': values['selling_price'] * (
                                1 + np.random.uniform(price_growth_rate['食用菌'][0], price_growth_rate['食用菌'][1])),
                    'cost_per_acre': values['cost_per_acre'] * (
                                1 + np.random.uniform(cost_growth_rate, cost_growth_rate)),
                }
            scenarios.append(temp_data)
        return scenarios


# 生成情景
scenarios = generate_scenarios(simulated_data, range(2024, 2031))

# 初始化 DEAP 环境
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
num_plots = sum([len(plots) for plots in land_areas.values()])
num_years = len(scenarios)
num_crops = len(valid_crops)
individual_length = num_plots * num_years * num_crops

toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=individual_length)

# 计算个体长度
num_plots = sum([len(plots) for plots in land_areas.values()])
num_years = len(scenarios)
num_crops = len(valid_crops)
individual_length = num_plots * num_years * num_crops

toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=individual_length)


# 检查约束条件
def check_constraints(decoded_individual, land_areas, valid_crops, actual_planting_2023):
    # 耕地面积限制
    for year in range(len(scenarios)):
        # 创建一个字典来存储每个地块的总种植面积
        plot_total_areas = {}

        for land_type, plots in land_areas.items():
```

```
            for plot, area in plots.items():
                # 计算每个地块的总种植面积
                if plot not in plot_total_areas:
                    plot_total_areas[plot] = 0

                for crop in valid_crops:
                    plot_total_areas[plot] += decoded_individual.get((land_type, plot, crop, year), 0)

                # 检查是否超出面积限制
                if plot_total_areas[plot] > area:
                    return False

    # 不重茬约束
    crops = [None] * len(scenarios)
    for year in range(len(scenarios)):
        for crop in valid_crops:
            if decoded_individual[(land_type, plot, crop, year)]:
                crops[year] = crop
                break
        for year in range(len(scenarios) - 1):
            if crops[year] == crops[year + 1]:
                return False

        # 豆类作物三年内种植一次的约束
    legume_years = [year for year in range(len(scenarios)) if
                    any(valid_crops[crop] for crop in valid_crops if decoded_individual[(land_type, plot, crop, year)])]
    if len(legume_years) > 0 and max(legume_years) - min(legume_years) > 2:
        return False

    # 每种作物每季的种植地不能太分散
    for crop in valid_crops:
        for year in range(len(scenarios)):
            plots_per_type = {}
            for land_type, plots in land_areas.items():
                planted_plots = [plot for plot in plots if decoded_individual[(land_type, plot, crop, year)]]
                if planted_plots:
                    plots_per_type[land_type] = planted_plots

            for land_type, planted_plots in plots_per_type.items():
                if len(planted_plots) > 1:
                    if not is_consecutive(planted_plots):
                        return False
    return True

```

```
285.
286.     # 适应度函数
287.
288.     def evaluate(individual, scenarios, land_areas, valid_crops, actual_planting_2023, substitute_complement):
289.         total_profit = 0
290.         decoded_individual = decode_individual(individual, num_years, land_areas, valid_crops)
291.
292.         # 检查约束条件
293.         if not check_constraints(decoded_individual, land_areas, valid_crops, actual_planting_2023):
294.             return -1,
295.
296.         # 计算总利润，考虑作物间的替代性和互补性
297.         for year in range(len(scenarios)):
298.             for land_type, plots in land_areas.items():
299.                 for plot, area in plots.items():
300.                     for crop1, data1 in scenarios[year].items():
301.                         for crop2, data2 in scenarios[year].items():
302.                             if (crop1, crop2) in substitute_complement:
303.                                 relation = substitute_complement[(crop1, crop2)]
304.                                 if decoded_individual[(land_type, plot, crop1, year)] > 0 and decoded_individual[(land_type, plot, crop2, yea
    r)] > 0:
305.                                     if relation > 0:  # 互补性
306.                                         total_profit += relation * data1['yield_per_acre'] * data1['selling_price']
307.                                     else:  # 替代性
308.                                         total_profit -= abs(relation) * data1['cost_per_acre']
309.
310.         return total_profit,
311.
312.     # 解码个体
313.     def decode_individual(individual, num_years, land_areas, valid_crops):
314.         decoded_individual = {}
315.         index = 0
316.         for year in range(num_years):
317.             for land_type, plots in land_areas.items():
318.                 for plot in plots:
319.                     for crop in valid_crops:
320.                         # 假设每个基因代表作物在地块上的种植比例
321.                         decoded_individual[(land_type, plot, crop, year)] = individual[index] / len(valid_crops) * \
322.                                                                             land_areas[land_type][plot]
323.                         index += 1
324.         return decoded_individual
325.
326.     # 注册评估函数
327.     toolbox.register("evaluate", functools.partial(evaluate, scenarios=scenarios, land_areas=land_areas, valid_crops=valid_crops,
```

```
                    actual_planting_2023=actual_planting_2023, substitute_complement=substitute_complement_df))

# 判断地块名称是否连续
def is_consecutive(plots):
    plots_sorted = sorted([int(plot[1:]) for plot in plots])
    return all(plots_sorted[i] + 1 == plots_sorted[i + 1] for i in range(len(plots_sorted) - 1))

# 辅助函数，用于传递固定参数
def evaluate_wrapper(ind):
    return evaluate(ind, scenarios, land_areas, valid_crops, actual_planting_2023, substitute_complement)

# 注册评估函数
toolbox.register("evaluate", evaluate_wrapper)

# 交叉操作
toolbox.register("mate", tools.cxTwoPoint)
# 变异操作
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
# 选择操作
toolbox.register("select", tools.selTournament, tournsize=3)




# 初始化种群
POPULATION_SIZE = 100
HALL_OF_FAME_SIZE = 1
population = [toolbox.individual() for _ in range(POPULATION_SIZE)]
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

# 进化过程
NGEN = 50
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("avg", np.mean)
stats.register("std", np.std)
stats.register("min", np.min)
stats.register("max", np.max)

population, logbook = algorithms.eaSimple(population, toolbox, cxpb=0.5, mutpb=0.2, ngen=NGEN, stats=stats,
                                          halloffame=hof, verbose=True)

# 输出最优解
best_individual = hof[0]
decoded_best_individual = decode_individual(best_individual, num_years, land_areas, valid_crops)

```

```
print("Best Individual Fitness:", best_individual.fitness.values)
print("Decoded Best Individual:")

# 提取最优解中的具体种植面积信息
results = []
for year in range(len(scenarios)):
    for land_type, plots in land_areas.items():
        for plot in plots:
            for crop in valid_crops:
                # 获取特定年份、土地类型、地块和作物的种植面积
                planting_area = decoded_best_individual.get((land_type, plot, crop, year), 0)
                if planting_area > 0:
                    results.append({
                        '年份': 2024 + year,
                        '地块类型': land_type,
                        '地块名称': plot,
                        '作物名称': crop,
                        '种植面积': planting_area
                    })

# 保存结果到Excel
output_path = r'C:\Users\HP\Desktop\种植方案 3_1.xlsx'
output_df = pd.DataFrame(results)
with pd.ExcelWriter(output_path) as writer:
    output_df.to_excel(writer, sheet_name='种植方案', index=False)

print(f"Results saved to {output_path}")
```