杭电数学建模竞赛

摘　要

柔性版印刷目前被广泛认为是印刷行业中公认的绿色印刷方式之一。在我国，柔性版印刷广泛应用于瓦楞纸箱、标签、无菌液体包装、纸杯纸袋、餐巾纸等领域，并逐步占据主导地位。由于大型印刷机的插槽数量有限，无法一次性将所有的墨盒全部放置在插槽中，印刷过程中需要频繁更换墨盒，严重影响了印刷效率。因此，通过**整数规划方法**、**剪枝优化方法**以及**贪心-模拟退火算法**来减小总切换时间，可以显著提升印刷效率。

针对问题一，为了实现总切换次数的最小化，建立了一个**整数规划模型**。模型中引入了一个主要决策变量，用于表示相邻包装种类之间的墨盒切换情况。目标函数是最小化所有墨盒切换的总次数，通过约束条件确保每个包装种类的墨盒需求得到满足，同时每个插槽只能容纳一个墨盒。利用**剪枝优化的分支定界法**求解该模型并找到最优解，从而最小化总的墨盒切换次数。根据附件一的数据，最终求得的最小切换次数，分别为**表一 5 次**、**表二 8 次**、**表三 48 次**、**表四 58 次**和**表五 108 次**。详细结果见第四章第一节。

针对问题二，在问题一的基础上，引入不同墨盒之间的切换时间，增加一个切换时间矩阵。模型需要一个主要决策变量和一个辅助决策变量，用于表示相邻包装种类之间的墨盒切换情况以及相邻包装种类之间的切换时间。目标函数是最小化所有墨盒切换的总切换时间，通过约束条件确保每个包装种类的墨盒需求得到满足。通过**剪枝优化的分支定界法**求解该模型并找到最优解，从而最小化总的墨盒切换时间。根据附件二的数据，最终求得的最小切换时间，分别为**表一 32 分钟**、**表二 51 分钟**、**表三 132 分钟**、**表四 258 分钟**和**表五 448 分钟**。详细结果见第四章第二节。

针对问题三，为了确保墨盒按照前后相对顺序放置，在数学模型中增加了一个墨盒相对位置约束，确保每个包装种类的墨盒按照给定的顺序排列在插槽中。通过建立一个整数规划模型，目标是最小化总切换时间。定义决策变量和辅助变量，并增加约束条件，在给定包装种类印刷顺序的情况下，使墨盒总切换时间最小。根据附件三的数据，最终求得的最小切换时间，分别为**表一 56 分钟**、**表二 189 分钟**、**表三 239 分钟**和**表四 288 分钟**。详细结果见第四章第三节。

针对问题四，在印刷之前需要确定包装种类印刷顺序的情况下，提出了一种结合贪心算法和模拟退火算法的方法。**贪心算法用于计算局部最优解**，而模拟退火算法通过接受一定概率的次优解来跳出局部最优，从而找到**全局最优解**。通过建立**动态规划模型**，利用贪心算法计算在约束条件下的局部最优解，由于模拟退火算法有一定概率接受比当前解差的解，可以跳出局部最优解，从而达到全局最优解。根据附件四的数据，最终求得的最小切换时间，分别为**表一 21 分钟**、**表二 37 分钟**、**表三 71 分钟**和**表四 332 分钟**。详细结果见第四章第四节。

针对问题五，相较于问题四的场景，增加了一台喷雾清洗机，因此需要考虑如何协调两台清洗机的工作，以最小化总切换时间。建立一个**混合整数线性规划**

模型，并结合**贪心算法与元启发式算法**求解该模型，最终确定最优包装印刷顺序，求得最小墨盒总切换时间。根据附件五的数据，最终求得的最小切换时间，分别为**表一 16 分钟**、**表二 168 分钟**、**表三 219 分钟**和**表四 239 分钟**。详细结果见第四章第五节。

综上所述，本文所使用的贪心-模拟退火算法在**计算复杂度上相对较低**，且能够接近最优解。通过贪心算法快速生成初始解，确保了计算的高效性和可行性。模拟退火算法通过接受概率机制逐步优化解，在寻找全局最优解时，能够有效避免陷入局部最优。本文结合这两种算法的优势，既保留了贪心算法的高效性，又利用模拟退火算法的全局搜索能力，使得算法在**保证计算速度**的同时，能够接近最优解。这种组合方法不仅适用于单清洗机条件下的墨盒切换问题，还在多清洗机场景中中展现出较高的**实用性和有效性**，大大提升了计算效率和解决问题的能力。贪心-模拟退火算法的计算复杂度显著低于传统的全局搜索算法，适用于处理大规模优化问题，且已经在实际应用中展现出接近最优解的效果。

关键词: 线性规划，整数规划，剪枝优化，贪心算法，模拟退火算法，元启发式算法

一　问题背景

柔性版印刷目前是印刷行业公认的绿色印刷方式之一。

柔性版印刷最初被称之为苯胺印刷，起源于二十年代初期的美国，因其使用的苯胺染料油墨有毒，而没有得到发展，此后，油墨制造厂开始使用大家公认且可以接受的色料药剂，于 1952 年在美国第十四届包装研讨会上被更为名柔性版印刷 (FLEXOGRAPHIC PROCESS)，在台湾也被翻译为富瑞凸版印刷。七十年代中期以后，由于材料工业的进步，特别是高分子树脂版材和金属陶瓷网纹辊的问世，促使柔性版印刷的发展有了质的飞跃。在世界范围内成为增长速度最快的印刷方式，特别是在美国，得到了充分的发展，无论是印刷机的制造技术还是柔印的应用技术都代表了柔性版印刷的最高水平。美国国内市场所见到的大部分商品包装几乎都是柔性版印刷产品。

柔性版印刷主要特点包括：适用性广泛：柔性版印刷可以在多种材料上进行印刷; 高速印刷：柔性版印刷机能够以高速运行，适合大批量生产; 印刷质量好; 环保优势：柔性版印刷使用的油墨多为水基油墨或 UV 油墨，这些油墨相对传统的溶剂型油墨更环保，减少了挥发性有机化合物（VOC）的排放; 工艺简单; 灵活性强：柔性版印刷可以根据不同的印刷需求进行灵活调整，适合多种规格和样式的印刷任务; 适应性强：柔性版印刷设备可以很容易地进行调整和维护，适应不同厚度和质地的印刷材料。

我国的柔性版印刷在瓦楞纸箱、标签、无菌液体包装、纸杯纸袋、餐巾纸等领域应用广泛，并逐步占据主导地位。柔性版印刷通过采用电油墨的增材沉积的方式取代叠层铜的减材去除，具有减少浪费和快速原型设计等多种优势。但由于大型印刷机插槽数量有限，无法一次性将所有的墨盒全部放置在插槽中，**在印刷过程中需频繁更换墨盒，严重影响印刷效率**，因此通过减小总切换时间则可以大幅提升印刷效率。

二　问题分析

印刷机上有多个插槽，用来放置墨盒（一个墨盒对应一种颜色），插槽按照前后顺序排列，一个插槽对应一套传墨辊（墨斗辊和网纹辊）和滚筒。由于插槽数量有限，因此无法一次性将所有的墨盒全部放置在插槽中。在对一个种类的包装印刷时放置在插槽中的墨盒集合需要包含该包装所需的墨盒。但在印刷某种包装时即使除该包装所需的墨盒外，还存在其他墨盒放置在插槽中，也不影响该包装的正常印刷。在更换印刷种类时需更换放置在卡槽中的墨盒，并且对传墨辊和滚筒进行清洗，此类过程消耗的时间称为**切换时间**。针对某个插槽来说将放置在该插槽中的墨盒取出，将另一个墨盒放入该插槽中的操作称为一次**切换操作**。印刷过程中不同颜色墨盒之间的切换时间可能不同。传墨辊和滚筒采用喷雾清洗机进行清洗，一台喷雾清洗机一次只能清洗一套传墨辊和滚筒。当只有一台喷雾清洗机时，如果在印刷过程中频繁地更换墨盒，则会大幅增加印刷时间，这会严重影响印刷效率，因此通过减小总切换时间则可以大幅提升印刷效率。

综上所述，柔性版印刷需要综合考虑切换时间、切换操作、清洗过程等因素对印刷效率的影响，通过结合相关的文献资料，通过合理的假设，建立适当的数学模型，解决总切换次数最小化的问题、在多种约束条件的情况下总切换时间最小化的问题

问题一的分析: 墨盒顺序可以任意放置而不影响印刷效果，且**给定包装种类印刷顺序**的情况下，实现总切换次数最小化，

在柔性印刷生产过程中，墨盒的切换是一个关键环节，因为每次切换都会消耗时间和资源。因此，为了提高生产效率和降低成本，我们需要尽量减少墨盒的切换次数。问题一的核心目标是最小化总的墨盒切换次数。在给定的包装种类及其所需墨盒的情况下，我们需要合理安排墨盒的使用顺序，使相邻包装种类之间的墨盒切换次数最少。

通过**整数规划求解器**, 我们可以求解该模型并找到最优解，从而最小化总的墨盒切换次数。最终的输出结果包括每个包装种类使用的墨盒以及总的切换次数。这不仅有助于优化生产流程还能显著降低生产成本，提高生产效率，具体思想如图 2-1所示。

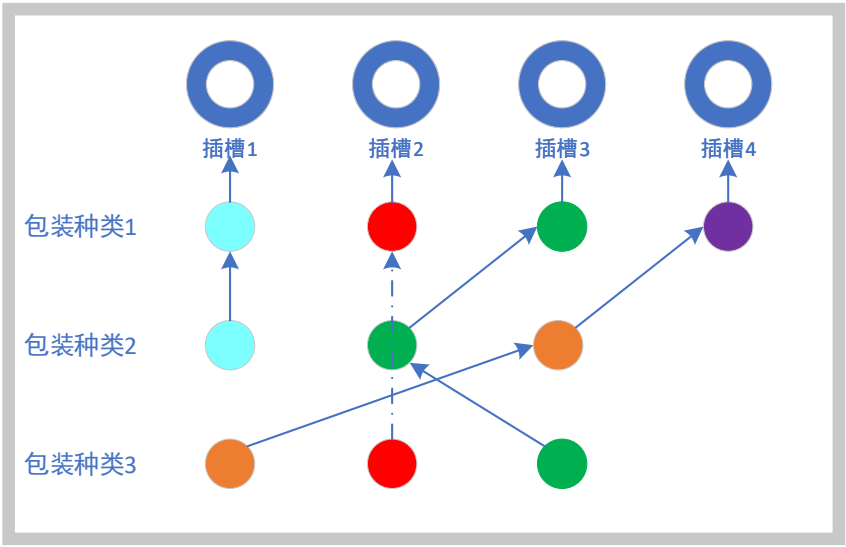

图 2-1: 问题一示意图

问题二的分析: 在问题一的约束条件下，引入**墨盒之间切换时间**概念（即，墨盒之间的切换时间示例表如下，切换时间单位为分钟（min）：例如，将放置在插槽中的墨盒 1 切换为墨盒 2 的切换时间为 10；将放置在插槽中的墨盒 2 切换为墨盒 1 的切换时间为 12），不同颜色墨盒之间的切换时间不完全相同，每种包装对应的墨盒顺序可以任意放置而不影响印刷效果。在给定包装种类印刷顺序的情况下，如何使总切换时间最小，具体思想如图 2-2所示。

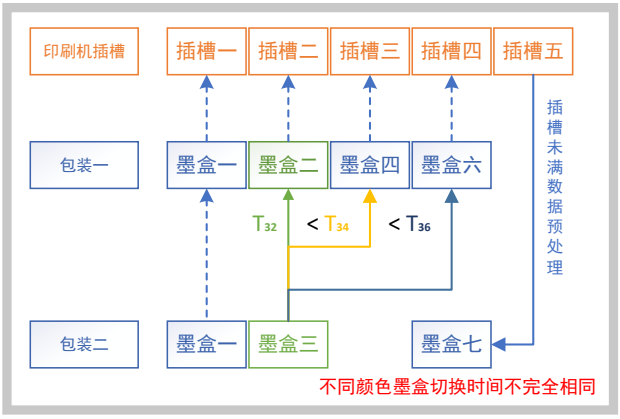

图 2-2: 问题二示意图

问题三的分析: 在问题二的约束条件下，引入**墨盒顺序**概念 (例如，柔性印刷机上有 4 个插槽，插槽序号分别为 1，2，3，4，并且按照序号从小到大的顺序前后排列。某包装所需的墨盒编号分别为 5，3，8，并且这三个墨盒需要按照 5 在 3 之前，3 在 8 之前的顺序放置在插槽中。墨盒放置情况即可以按照墨盒 5 放置在插槽 1 中，墨盒 3 放置在插槽 2 中，墨盒 8 放置在插槽 3 中的顺序放置；也可以按照墨盒 5 放置在插槽 1 中，墨盒 3 放置在插槽 2 中，墨盒 8 放置在插槽 4 中的顺序放置)，每种包装对应的墨盒顺序是不同且固定的，在给定包装种类印刷顺序的情况下，如何使总切换时间最小，具体思想如图 2-3所示。

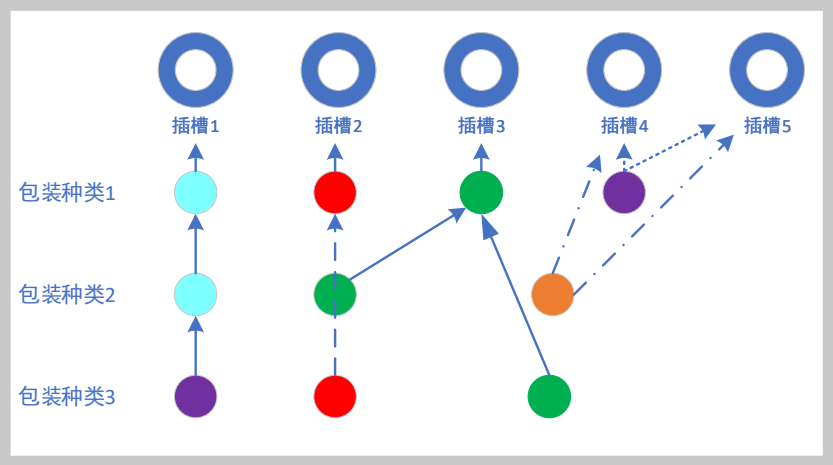

图 2-3: 问题三示意图

问题四的分析: 在问题三的约束条件下，在印刷之前**包装种类印刷顺序需要确定**的情况下 (包装种类不同，所需要的墨盒不同，导致的切换时间和切换操作不同)，如何使总切换时间最小。具体而言，给定不同包装对应的墨盒顺序，且不同颜色墨盒之间的切换时间不完全相同，我们需要找到一种包装印刷顺序，使得总切换时间最小化。这一问题涉及到排列组合和优化问题，是一个典型的 NP-hard 问题。每种包装有固定的墨盒顺序，定义了包装种类的印刷顺序后，可以确定每次切换的墨盒。主要变量是包装种类的顺序，约束条件为每个包装种类必须被印刷一次，墨盒切换时间依赖于前一个包装的墨盒顺序和当前包装的墨盒顺序。由于问题复杂，穷举法不可行。我们可以使用启发式算法来近似求解，如模拟退火算法。模拟退火算法通过随机交换包装顺序，计算新的总切换时间，并根据温度逐渐降低的原则决定是否接受新的解，从而避免陷入局部最优解。具体思想如图 2-4所示。

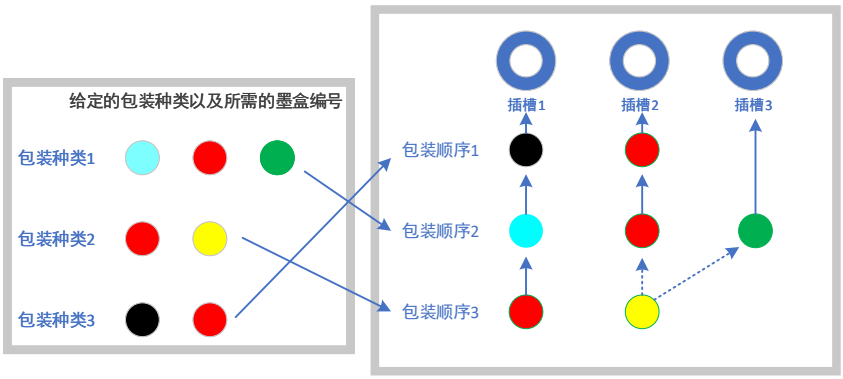

图 2-4: 问题四示意图

问题五的分析: 在问题四的约束条件下，增加了一台喷雾清洗机，一台喷雾清洗机一次只能清洗一套传墨辊和滚筒，因此有两台喷雾清洗机可以同时工作。需要协调两台喷雾清洗机的工作，以实现任务的合理分配。需要处理**并行调度问题**，确保两台清洗机的工作不会冲突，最大化利用两台设备的效率。目标仍然是最小化总切换时间，但需要考虑如何将任务分配给两台清洗机以实现这一目标。因此我们将该问题建立一个混合整数线性规划方法，并运用**贪心算法与元启发式算法**求解该模型，最终确定最优包装印刷顺序，求得最小墨盒总切换时间。具体思想如图 2-5所示。

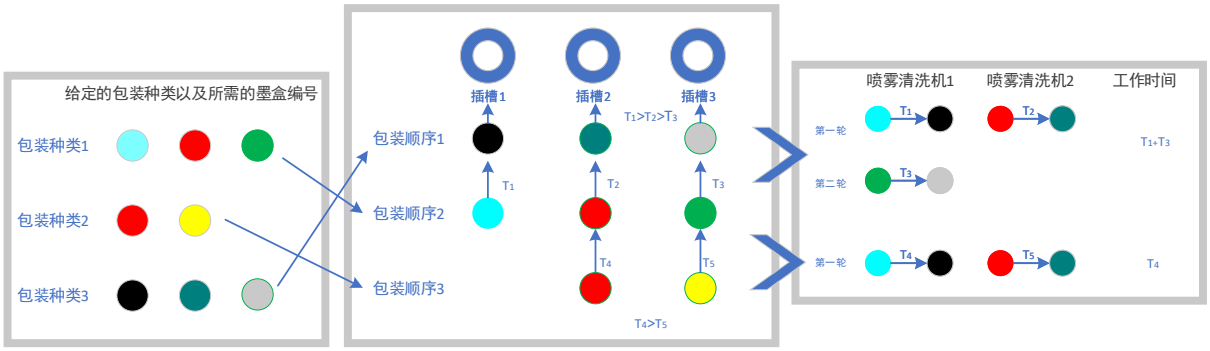

图 2-5: 问题五示意图

三　假设与符号说明

3.1　模型假设

(1) 墨盒固定顺序假设：每种包装对应的墨盒顺序是固定的，不会在不同印刷批次中变化。

(2) 单次切换时间假设：墨盒之间的切换时间是已知且固定的，即每个插槽从一种墨盒切换到另一种墨盒所需的时间是固定值。

(3) 并行清洗假设：两台清洗机可以同时并行工作，并且每台清洗机在处理一个插槽时，另一台清洗机可以独立处理其他插槽。

(4) 连续印刷假设：各种包装的印刷顺序在整个过程中是连续进行的，没有间断，即印刷一个包装后立即开始下一个包装的印刷。

(5) 互不干扰假设：两台清洗机在同一时刻处理不同的插槽，不会互相干扰，即一台清洗机清洗某个插槽时，另一台清洗机不会处理该插槽。

(6) 插槽容量假设：每个插槽只能容纳一种墨盒，不存在多个墨盒同时占用一个插槽的情况。

(7) 清洗时间优先假设：在计算总切换时间时，以两台清洗机中用时较长的清洗机的时间作为该次切换的时间。

(8) 墨盒可用性假设：所有需要的墨盒在印刷过程中是随时可用的，不存在墨盒短缺或等待补充的情况。

3.2 符号说明

模型符号	符号说明
M_p	第 p 种包装所需的墨盒集合
C	所有颜色墨盒集合
P	所有需印刷包装集合
t	包装印刷计数变量
p	包装印刷顺序编号为 t
S	印刷所有包装所需插槽集合
x_{ij}^k	第 k 个插槽内的墨盒 i 是否切换为墨盒 j
$x_{p,ij}^k$	包装 p 第 k 个插槽内的墨盒 i 是否切换为墨盒 j
y_i^k	墨盒 i 是否在插槽 k 中
$y_{i,t}^k$	第 t 次包装印刷墨盒 i 是否在插槽 k 中
$y_{i,p}^k$	包装 p 印刷时墨盒 i 是否在插槽 k 中
T_{ij}^k	插槽 k 中墨盒 i 切换为墨盒 j 的切换时间
$z_{p,i,j}^k$	包装 p 到 $p+1$ 时插槽 k 从墨盒是否由 i 切换到 j
r_m	第 p 个包装印刷所需的第 m 个墨盒
$\delta(\dot{})$	克罗内克函数
i	墨盒型号
j	墨盒型号
k	插槽型号
Z	总切换时间
I	总切换次数

四 模型建立

4.1 模型一

针对问题一，问题要求在给定包装种类印刷顺序的情况下，建立总切换次数最小化的数学模型，并根据附件 1 数据计算总切换次数，该问题的优化目标主要体现在对墨盒使用顺

序的规划上。

解决此模型主要用到整数线性规划相应知识，其基础知识如下：

线性规划方法作为运筹学中数学规划的一个重要分支，线性规划（Linear Programming ,LP）[1] 是一种研究线性约束条件下线性目标函数的极值问题的数学理论和方法。由于计算机能处理成千上万个约束条件和决策变量的 LP 问题，其已经成为现代管理中的一个基本方法。在实际工程中，我们需要将面临的问题归结成一个 LP 数学模型，其中关键点在于依据影响所要达到目的的因素选取适当的决策变量建立恰当的模型。描述 LP 问题的常用和最直观形式是标准型。标准型包括以下三个部分：

1. 目标函数：由 n 个决策变量 $\{x_i\}_{i=1}^n$ 与所达到目的之间的函数关系确定，并在一定的约束条件下使其最小或最大化。通常该函数是一个线性函数

$$\arg\min \sum_{i=1}^{n} c_i x_i,$$
$$\arg\max \sum_{i=1}^{n} c_i x_i,$$

   其中 $\{c_i\}_{i=1}^n$ 为目标函数系数。

2. 约束条件：决策变量 $\{x_i\}_{i=1}^n$ 必须满足的一组线性不等式或等式，其一般形式为

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &\leq b_1, \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &\leq b_n. \end{aligned}$$

   上述不等式组也可写为

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b},$$

   其中系数矩阵 $\boldsymbol{A}$ 由不等式组中的系数 $\{a_{ij}\}$ 组成。

3. 非负性约束：非负性约束要求决策变量 $\{x_i\}_{i=1}^n$ 非负，即 $x_i \geq 0$。

4.1.1 模型一建立

本题选择 0-1 整数线性规划模型[4] 求解，利用每个插槽内墨盒的切换确定相应的决策变量，从而建立所有包装的总切换次数目标函数，进一步通过恰当的约束条件对已建立的目标函数进行规划求解。模型建立如下：

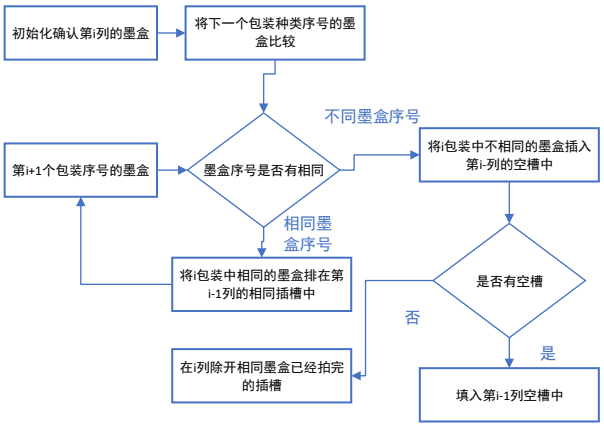

图 4-1: 问题一解决流程

1. **变量定义**：在给定包装种类印刷顺序的情况下，首先要定义适当的决策变量来进行建模；设 P 为需印刷包装集合；设 M_p 为第 p 种包装所需的墨盒集合；设 S 为插槽集合；设 x_{ij} 为二进制决策变量来表示第 j 个插槽内的第 i 个墨盒是否发生切换，即

$$x_i^k = \begin{cases} 1, & \text{当墨盒}k\text{在第}i\text{个插槽时}, \\ 0, & \text{当墨盒}k\text{不在第}i\text{个插槽时}. \end{cases} \quad \forall k \in S, \forall i \in C \tag{1}$$

2. **目标函数**：由题目可知，目标函数为最小化总切换次数，即插槽中墨盒的变更次数之和。通过克罗内克 δ 函数构造目标函数如下

$$\min \ \ I = \sum_{p \in P} \sum_{i \in C} \delta(x_i^k, x_i^{k+1}), \tag{2}$$

当在加工第 p 个和第 $p+1$ 个包装时，第 k 个墨盒从第 i 个插槽切换到了其他插槽，即 $x_i^k = 1 \neq x_i^{k+1} = 0$，则 $\delta(x_i^k, x_i^{k+1}) = 1$，表示进行了一次切换并累加次数；若第 k 个插槽内的墨盒未切换，即 $x_i^k = x_i^{k+1} = 1$，则 $\delta(x_i^k, x_i^{k+1}) = 0$，不进行累加。

3. **约束条件**：由上述决策变量所受的限制条件可以确定决策变量所要满足的约束条件。

- 显然，在每个插槽中最多放置有一个墨盒，即

$$\sum_{i \in C} x_i^k \leq 1, \quad \forall k \in S, \forall i \in C \tag{3}$$

- 每个包装种类的每个墨盒必须被分配到一个插槽

$$x_i^k \in M_p, \quad \forall k \in S, \forall i \in C \tag{4}$$

- 被剪枝优化算法剪掉的‘枝’无法再次进入对于二进制变量，假设被剪枝掉的解

x^* 满足 $x_1^* = 1, x_2^* = 0, \ldots, x_n^* = 1$，则排除约束为：

$$\sum_{i \in C}(x_i^k) * (1 - x_i^k) + (1 - (x_i^k)*)x_i^k \geq 1 \quad \forall k \in S, \forall i \in C \tag{5}$$

- 切换次数不大于插槽数与包装数的乘积

$$\sum_{i \in C} s_i \leq M_p \times S \quad \forall k \in S, \forall i \in C \tag{6}$$

- 每种墨盒只能分配到一个插槽

$$\sum_{i \in C} x_i^k = 1 \quad \forall k \in S, \forall i \in C \tag{7}$$

如上，我们将问题 1 转化为了等价的 0-1 型整数线性规划问题：

$$\min \ \ I = \min \ \ I = \sum_{p \in P} \sum_{i \in C} \delta(x_i^k, x_i^{k+1}) \quad \forall k \in S, \forall i \in C$$

$$s.t. \begin{cases} \sum_{i \in C} x_i^k \leq 1, \quad \forall k \in S, \forall i \in C \\ x_i^k \in M_p, \quad \forall k \in S, \forall i \in C \\ x_i^k \in \{0, 1\}, \quad \forall k \in S, \forall i \in C \\ \sum_{i \in C}(x_i^k) * (1 - x_i^k) + (1 - (x_i^k)*)x_i^k \geq 1 \quad \forall k \in S, \forall i \in C \\ \sum_{i \in C} s_i \leq M_p \times S \quad \forall k \in S, \forall i \in C \\ \sum_{i \in C} x_i^k = 1 \quad \forall k \in S, \forall i \in C \end{cases}$$

4.1.2 模型一数据预处理

数据清洗

去除在' 包装种类编号'、' 墨盒编号' 和' 插槽序号' 列中有缺失值的行，确保后续处理的数据是完整的，没有缺失的必要信息。

提取唯一值

提取数据中的唯一包装种类编号和墨盒编号, 用于构建包装类型与墨盒编号的映射和约束条件。

确定插槽数量

找到插槽序号的最大值。

矩阵预填充

为方便后续计算，且经过测试不会对实验结果产生不良影响，本文在数据预处理时将包装种类-插槽矩阵第一行填充至最大插槽数。如图 4-2所示，取问题一附件第三张表前四行数据为例，本章第三节求解结果时将展示此预处理的优势。

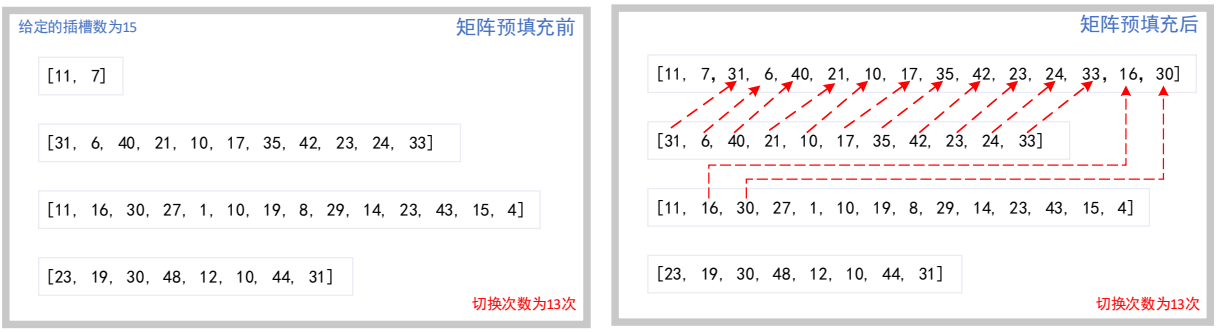

图 4-2: 矩阵预处理示意图

4.1.3 模型一求解

(1) 算法优化-剪枝优化

本文通过剪枝策略优化给出的整数线性规划算法，通过不断分枝、定界和剪枝求解墨盒切换的全局最优顺序。

在搜索算法中优化中，剪枝，就是通过某种判断，避免一些不必要的遍历过程，形象的说，就是剪去了搜索树中的某些“枝条”，故称剪枝。应用剪枝优化的核心问题是设计剪枝判断方法，即确定哪些枝条应当舍弃，哪些枝条应当保留的方法。

剪枝优化三原则: 正确、准确、高效. 原则。搜索算法, 绝大部分需要用到剪枝. 然而, 不是所有的枝条都可以剪掉, 这就需要通过设计出合理的判断方法, 以决定某一分支的取舍. 在设计判断方法的时候, 需要遵循一定的原则.

1）正确性

枝条不是爱剪就能剪的. 如果随便剪枝, 把带有最优解的那一分支也剪掉了的话, 剪枝也就失去了意义. 所以, 剪枝的前提是一定要保证不丢失正确的结果.

2）准确性

在保证了正确性的基础上, 我们应该根据具体问题具体分析, 采用合适的判断手段, 使不包含最优解的枝条尽可能多的被剪去, 以达到程序“最优化”的目的. 可以说, 剪枝的准确性, 是衡量一个优化算法好坏的标准.

3）高效性

设计优化程序的根本目的, 是要减少搜索的次数, 使程序运行的时间减少.

对于两个满插槽的包装种类如果存在相同元素，那么根据剪枝策略必然会减去一条无用的支路，如图 4-3剪枝类型 1 中，因有一种情况第二卡槽皆为 5，另一情况则无相同元素在同一卡槽，则剪去；对于一个满插槽一个非满插槽的包装种类，则需根据下一种包装种类的插槽类型来进行剪枝，如剪枝类型 2 中，因第四层与第二层有 5 这个相同元素，且在第二层中 5 处于第二卡槽，故将单卡槽但位于第二卡槽的第一种情况剪去。

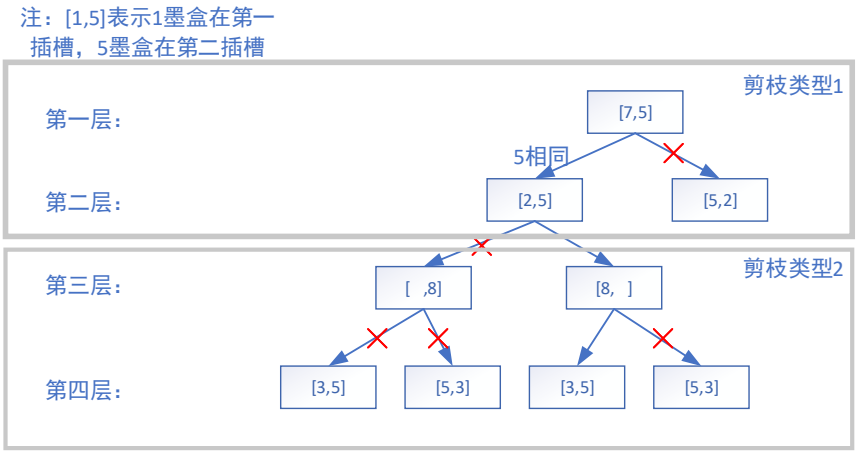

图 4-3: 剪枝优化算法示意图

(2) 算法分析

整数线性规划模型中定义的约束条件包括：每个包装种类的墨盒需求必须得到满足，每个插槽只能容纳一个墨盒，辅助变量 $y_{p,i,j}$ 确保模型能够正确反映墨盒切换的实际情况。通过这些约束条件，模型能够在保证每种包装都能顺利印刷的前提下，尽量减少墨盒的切换次数。通过**整数规划求解器**, 我们可以求解该模型并找到最优解，从而最小化总的墨盒切换次数。但因其仍属于暴力求解范畴，计算复杂度高，运行时间长，故提出一种剪枝优化策略，将提高计算复杂度增加算法时间的多余的枝剪去，最终的输出结果包括每个包装种类使用的墨盒以及总的切换次数。这不仅有助于优化生产流程还能显著降低生产成本，提高生产效率。

(2) 算法流程

针对提出的数学模型，本文提出了一种改进的分支定界算法。在进行墨盒切换时，比较前后两包装印刷所需的墨盒集合以及当前包装所需墨盒集合，根据不同的剪枝策略进行优化并得到全局最优的切换顺序。

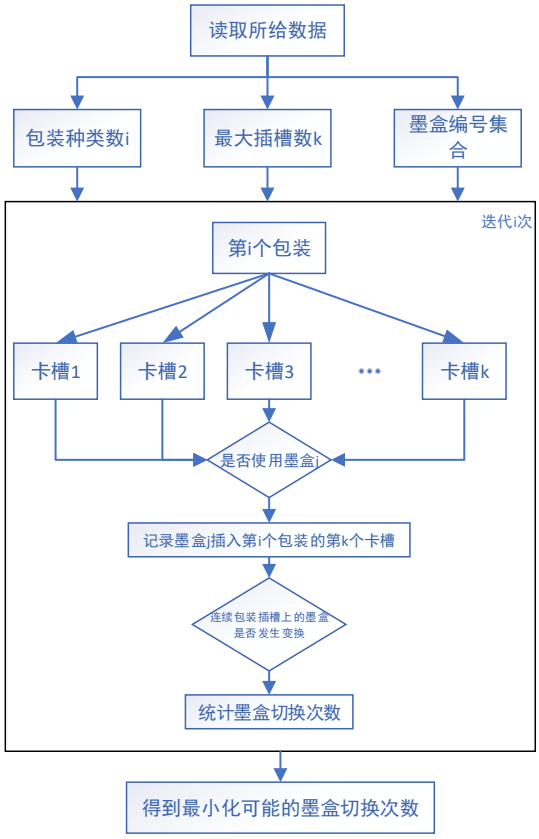

图 4-4: 问题一解决流程图

4.1.4 模型一结果分析

根据上述方法，对问题一建立的模型进行求解后，得出最优结果，如下所示。

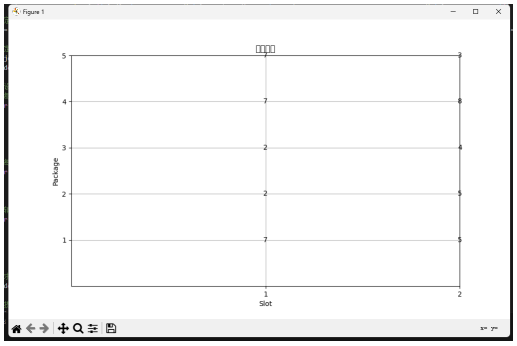

图 4-5: 表一数据结果图

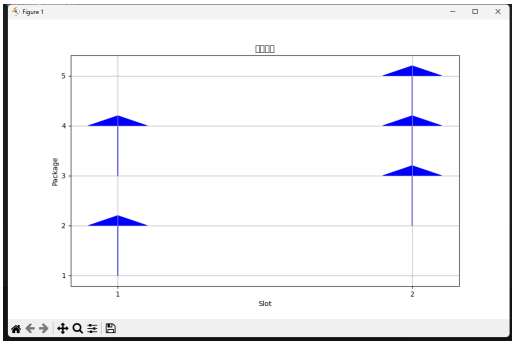

图 4-6: 表一可视化结果图

从上图所示，经过模型求解后，表一的最小化墨盒总切换次数为 **5** 次。

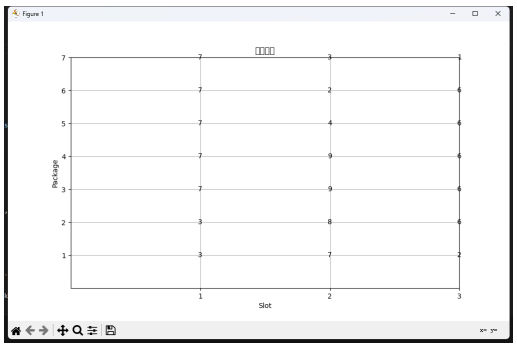

图 4-7: 表二数据结果图

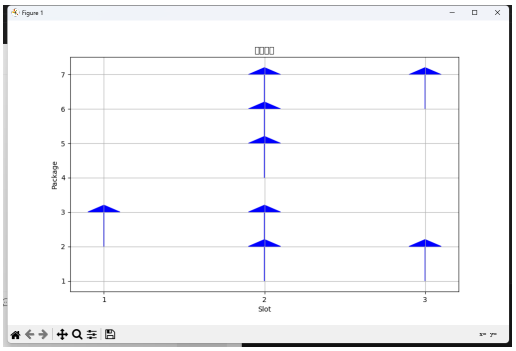

图 4-8: 表二可视化结果图

从上图所示，经过模型求解后，表二的最小化墨盒总切换次数为 **8** 次。

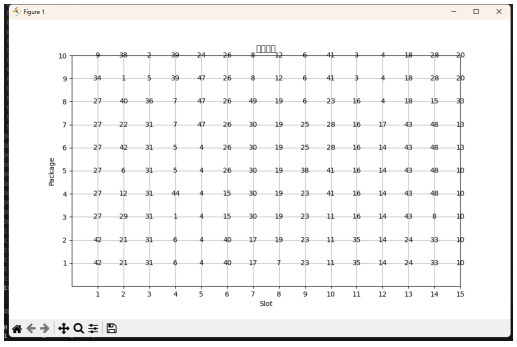

图 4-9: 表三数据结果图

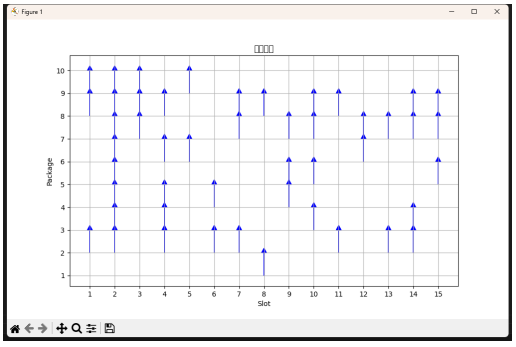

图 4-10: 表三可视化结果图

从上图所示，经过模型求解后，表三的最小化墨盒总切换次数为 **48** 次。

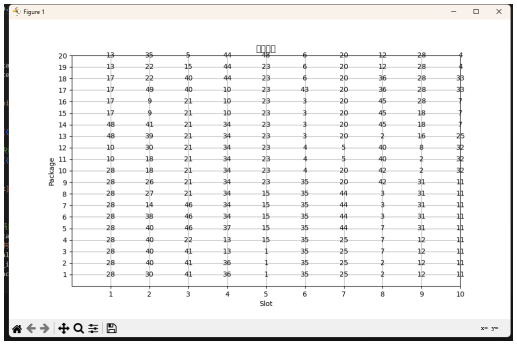

图 4-11: 表四数据结果图

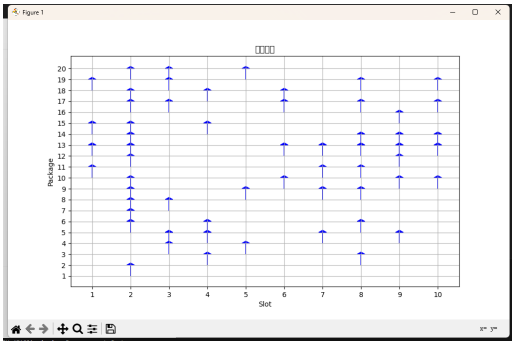

图 4-12: 表四可视化结果图

从上图所示，经过模型求解后，表四的最小化墨盒总切换次数为 **58** 次。

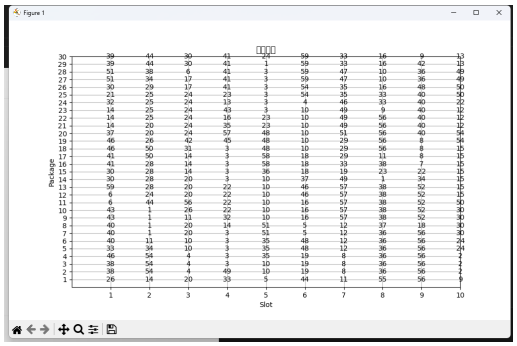

图 4-13: 表五数据结果图

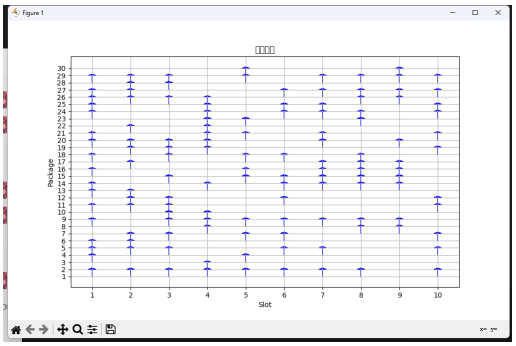

图 4-14: 表五可视化结果图

从上图所示，经过模型求解后，表一的最小化墨盒总切换次数为 **108** 次。

上述结果为根据本文改进的证书优化模型和优化方法得出的最优化结果，对比未曾经过优化的整数规划算法如下图所示 (以附件一的表三结果为例)，本文算法计算**精度更高，效率更高**。未曾经过矩阵预填充的算法第一行出现卡槽数个所要求的墨盒种类，由此可知，本文效果有明显改善。

```
Package 1 uses inkbox 11 in slot 2
Package 1 uses inkbox 11 in slot 3
Package 1 uses inkbox 11 in slot 4
Package 1 uses inkbox 11 in slot 5
Package 1 uses inkbox 11 in slot 7
Package 1 uses inkbox 35 in slot 4
Package 2 uses inkbox 6 in slot 2
Package 2 uses inkbox 11 in slot 3
Package 2 uses inkbox 11 in slot 4
Package 2 uses inkbox 11 in slot 5
Package 2 uses inkbox 11 in slot 6
Package 2 uses inkbox 11 in slot 7
Package 2 uses inkbox 17 in slot 10
Package 2 uses inkbox 24 in slot 7
Package 2 uses inkbox 31 in slot 3
Package 2 uses inkbox 35 in slot 4
Package 2 uses inkbox 42 in slot 6
Package 3 uses inkbox 1 in slot 3
Package 3 uses inkbox 8 in slot 6
```

图 4-15: 表三数据对比结果图

4.2 模型二

在问题二的基础上，我们引入了墨盒之间切换时间的概念。例如，不同颜色墨盒之间的切换时间各不相同，切换时间的示例表如下：将插槽中的墨盒 1 切换为墨盒 2 的时间为 10 分钟，而将墨盒 2 切换为墨盒 1 的时间为 12 分钟。在给定包装种类印刷顺序的情况下，我们的目标是使总切换时间最小化。

为了解决这一问题，我们延续了问题一中使用的模型，并增加了一个变量 T_{ij}^{k}，表示从墨盒 i 切换到墨盒 j 所需要的时间。根据提供的数据，构建了一个整数规划模型，以优化生产过程中的墨盒切换时间。通过优化模型，找到一种墨盒切换的顺序，使总切换时间最小化。

在解决墨盒切换问题时，贪心算法[2, 3] 提供了一种快速且直观的方法来生成初始解。由于不同颜色墨盒之间的切换时间不同，而每种包装对应的墨盒顺序是固定的，因此在每次墨盒切换过程中，选择最优的切换方案至关重要。贪心算法在这种情况下的优势体现在以下几个方面：

首先，贪心算法能够迅速生成一个可行解。在每一步决策中，贪心算法选择当前状态下最优的操作，即选择切换时间最短的墨盒。这种局部最优的选择过程简单明了，计算量较小，能够在较短时间内生成一个初始解，为后续的优化算法提供基础。其次，贪心算法的策略易于实现和理解。通过每次选择最小切换时间的墨盒，贪心算法逐步构建整个墨盒切换过程。这种方法不需要复杂的数据结构或算法，仅需对当前状态进行简单判断和选择，因此易于实现，且在实践中具有较高的可靠性和稳定性。此外，虽然贪心算法本身可能无法保证找到全局最优解，但它生成的初始解通常具有较高质量。由于每次选择的是当前最优操作，贪心算法生成的解在一定程度上已经接近最优。将这个初始解作为起点，可以显著提升后续优化算法（如模拟退火算法）的效率和效果。

总之，贪心算法在墨盒切换问题中的应用，通过快速生成高质量初始解，简化了问题复杂性，并为进一步的全局优化打下了良好基础。这种方法不仅高效且易于实现，是解决墨盒切换问题的理想选择。

4.2.1 模型二建立

1. **变量定义**：设从颜色 i 的墨盒切换到颜色 j 的墨盒需要时间 T_{ij}^k；设 C 为墨盒颜色集合；设 S 为印刷机的插槽集合；设 P 为需要印刷的包装集合，t 为包装印刷计数变量。设 x_{ij}^k 为表示插槽 k 中是否由颜色 i 切换到颜色 j 的二进制变量；设 y_i^k 为表示颜色 i 是否在插槽 k 中，即

$$x_{ij}^k = \begin{cases} 1, & \text{插槽}k\text{中墨盒的颜色由}i\text{切换为}j, \\ 0, & \text{插槽}k\text{中墨盒的颜色并未切换.} \end{cases} i \in C, j \in C, k \in S \tag{8}$$

$$y_{i,t}^k = \begin{cases} 1, & \text{颜色为}i\text{的墨盒在插槽}k\text{中}, \\ 0, & \text{颜色为}i\text{的墨盒不在插槽}k\text{中.} \end{cases} i \in C, j \in C, k \in S \tag{9}$$

2. 由于在利用贪心算法求解最优切换顺序之前首先进行数据预处理，且插槽在插入墨盒之后只有不切换和切换两种状态，即该插槽始终保持放置有墨盒。因此只有在印刷第一个包装前进行算法初始化时会出现插槽未放置满的情况，随着印刷的进行，在当前包装印刷时放置有墨盒的插槽个数一定大于等于印刷上个包装时放置有墨盒的插槽个数。即

$$\sum_{k \in S} \sum_{i \in C} y_{i,t+1}^k \geq \sum_{k \in S} \sum_{i \in C} y_{i,t}^k \tag{10}$$

3. **约束条件**：

- 显然，与问题一相同，在每个插槽中最多只能放置有一种颜色的墨盒，即该插槽有放置墨盒或并未使用。该约束条件表示如下：

$$\sum_{i \in C} y_i^k \leq 1, \quad \forall k \in S \tag{11}$$

- 每个包装类型对应正确的颜色集合。对于所有的颜色 $j \in C$，在插槽 k 中是否由颜色 i 切换到其他颜色（包括不切换）应该与颜色为 i 的墨盒是否在插槽 k 中一

致，即若颜色为 i 的墨盒在插槽 k 中，则由所有颜色 $j \in C$ 切换到颜色 i 墨盒的情况不存在，对应的决策变量之和为 0，当且仅当不切换时为 1；若颜色为 i 的墨盒不在插槽 k 中，则由所有颜色 $j \in C$ 切换到颜色 i 墨盒（包括不切换）的情况可能存在也可能不存在，对应的决策变量之和为 0 或 1。该约束条件表示如下：

$$y_i^k \geq \sum_{j \in C} x_{ji}^k, \quad \forall k \in S, \forall i \in C \tag{12}$$

- 各个包装类型之间的颜色顺序一致。由题目分析可知，包装种类的印刷顺序固定且墨盒顺序可以任意放置而不影响印刷效果。因此，插槽 k 中颜色为 i 的墨盒是否切换到颜色为 j 的墨盒应该与颜色为 i 或 j 的墨盒是否在插槽 k 中一致。即若插槽 k 中墨盒的颜色由 i 切换到 j，对应的决策变量 x_{ij}^k 为 1，此时颜色为 i 的墨盒在插槽 k 中而颜色为 j 的墨盒不在插槽 k 中，对应的二元变量之差等于 1；若插槽 k 中墨盒的颜色不进行切换，对应的决策变量 x_{ij}^k 为 0，此时同样此时颜色为 i 的墨盒在插槽 k 中而颜色为 j 的墨盒不在插槽 k 中，对应的二元变量之差等于 1。该约束条件表示如下：

$$y_i^k - y_j^k \geq x_{ij}^k \quad \forall k \in S, \forall i \in C, \forall j \in C \tag{13}$$

如上，我们将问题二转化为了等价的 $0-1$ 整数线性规划模型：

$$\min \ Z = \sum_{k \in S} \sum_{i \in C} \sum_{j \in C} T_{ij}^k x_{ij}^k,$$

$$s.t. \begin{cases} \sum_{i \in C} y_i^k \leq 1, \quad \forall k \in S \\ y_i^k \geq \sum_{j \in C} x_{ji}^k, \quad \forall k \in S, \forall i \in C \\ y_i^k - y_j^k \geq x_{ij}^k, \quad \forall k \in S, \forall i \in C, \forall j \in C \\ \sum_{k \in S} \sum_{i \in C} y_{i,t+1}^k \geq \sum_{k \in S} \sum_{i \in C} y_{i,t}^k \quad \forall k \in S, \forall i \in C, \forall j \in C \end{cases}$$

4.2.2 模型二求解

通过对问题的分析可知，不同颜色墨盒的切换时间不同，且对问题二所给数据进行观察可以发现由颜色 i 切换到颜色 j 的时间与由颜色 j 切换到颜色 i 的时间也可能不同，因此一般的算法无法得到较优的结果。对于问题二模型的算法设计可首先通过设计的贪心算法求得一个可行解，之后再通过模拟退火的方法将初始解进行优化，从而实现总切换时间最小化的目标。

在问题一的基础上本文尝试使用通过分枝定界法对构造在约束条件下的的最优函数进行直接求解。但发现在附件 2 的表一和表二数据中均展现了较好的效果，但是当墨盒数量增大，算法运算量随着所对应的时间切换矩阵的维数增加而急剧增大，且难以准确的计算出最优的解。于是采用贪心算法，将整个墨盒排序策略问题分解成多个步骤进行，在每一个步骤都优先选取当前步骤的最优方案。直到所有步骤全部结束；在每一步都不考虑对后续步骤的影响，在后续步骤中也不再回过去改变先前的选择。由局部最优逐步推演到全局

最优。例如存在编号为 1,3 的墨盒, 要与下一个包装种类中的 1,2 号墨盒进行更换。

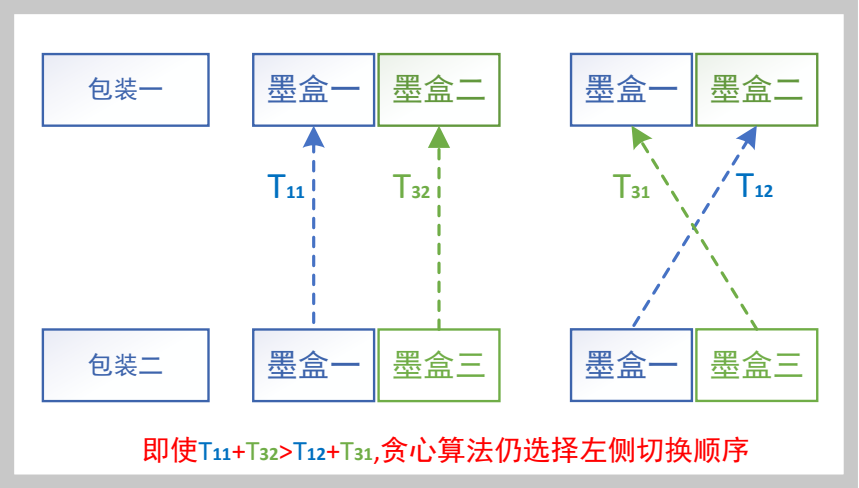

图 4-16: 贪心算法决策过程

(1) 贪心算法的设计：

本文所设计的贪心算法主要分为两个步骤，在所有包装的印刷过程中利用者两个步骤不断循环求解。第一个步骤是当执行一个包装的印刷前，分别比较所需墨盒的颜色与下一个待印刷包装所需墨盒的颜色以及所需的插槽个数。第二个步骤是比较当前需印刷包装与下一个待印刷包装需要的墨盒集合，分别计算不同切换顺序下的切换时间比较当前需印刷包装的墨盒与下一个待印刷包装墨盒不同的切换时间，并根据一定的规则对不同切换顺序进行筛选，从而逐步得到局部最优的墨盒切换顺序。

步骤一：墨盒切换的颜色比较与切换时间

在进行墨盒切换时，需要根据同一个插槽内当前需印刷包装与下一个待印刷包装需要的墨盒颜色是否相同以及所需插槽个数同时进行墨盒的切换选择。当所需插槽数小于最大插槽数时，由问题一求解方法中的数据预处理手段对墨盒顺序矩阵进行预填充，将当前需印刷包装未使用的插槽依照印刷顺序依次填入待印刷包装所需的颜色墨盒编号，减少切换次数从而减少切换时间。

步骤二：

方式一：当前需印刷包装所需的墨盒集合与下一个待印刷包装所需的墨盒集合中有相同的颜色墨盒时，对所有重合组合剩余不同墨盒进行不同组合并对相应的切换时间进行比较，按照切换时间最小原则筛选出一个切换顺序；之后按照有相同墨盒但不重合排列的规则得到新的切换顺序并按照切换时间最小原则筛选出一个切换顺序。最后将得到的两个切换顺序再次按照切换时间最小原则进行比较，筛选出当前包装的墨盒切换顺序。

图 4-17: 方式 2

方式二：当前需印刷包装所需的墨盒集合与下一个待印刷包装所需的墨盒集合中没有有相同的颜色墨盒时，在约束条件下对所有切换顺序可行解按照切换时间最小原则筛选出当前包装的墨盒切换顺序。

(1) 贪心算法流程图：问题二贪心算法流程图如下所示：

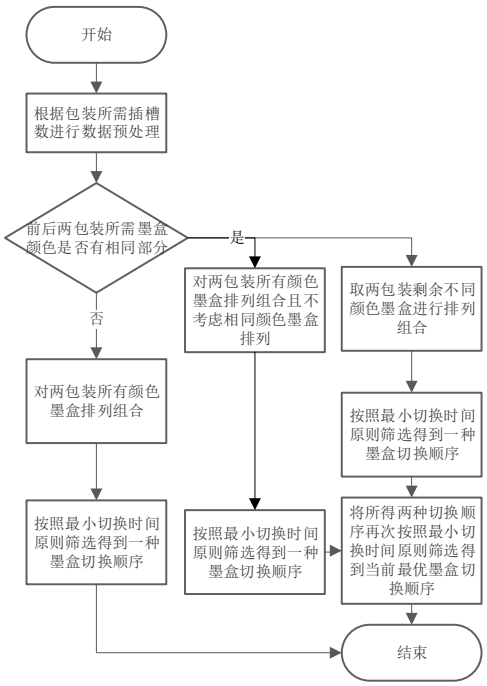

图 4-18: 模型二贪心算法流程图

4.2.3 模型二结果分析

根据所提出的算法计算得出，在问题二的约束条件下，墨盒总切换时间最短结果。

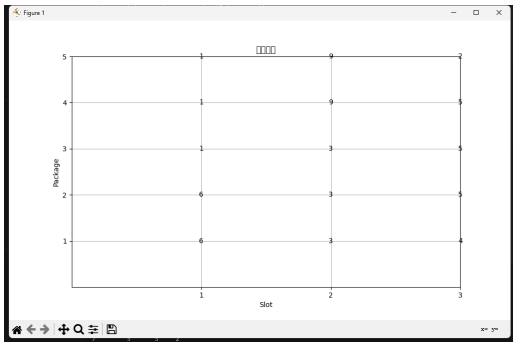

图 4-19: 表一数据结果图

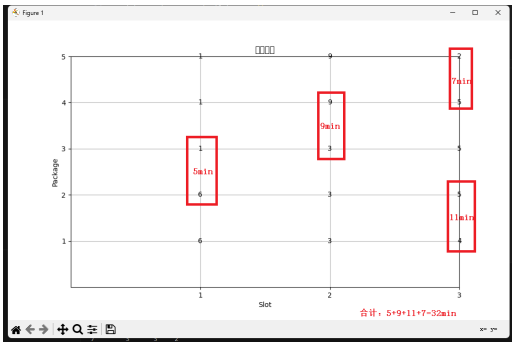

图 4-20: 表一切换时间计算示意

从上图所示，经过模型求解后，表一的最小化墨盒总切换时间为 **32(min)**。

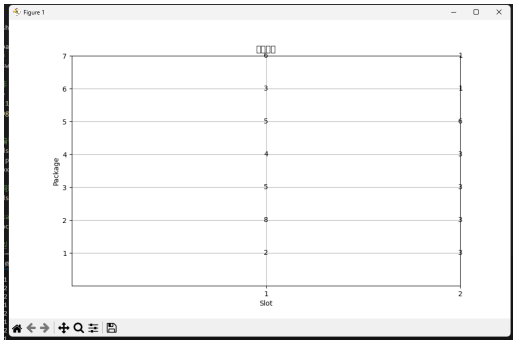

图 4-21: 表二数据结果图

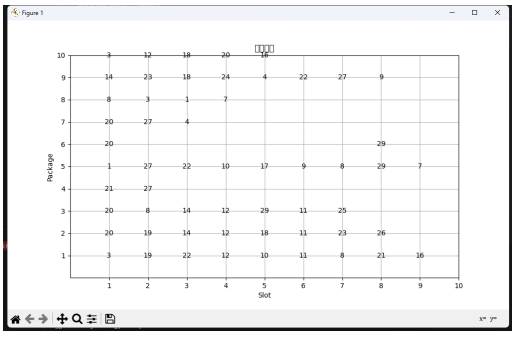

图 4-22: 表三数据结果图

从上图所示，经过模型求解后，表二的最小化墨盒总切换时间为 **51(min)**，表三的最小化墨盒总切换时间为 **132(min)**。

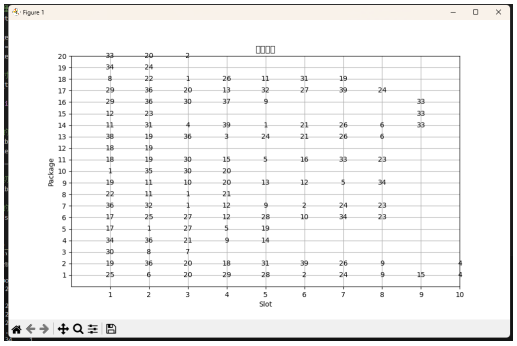

图 4-23: 表四数据结果图

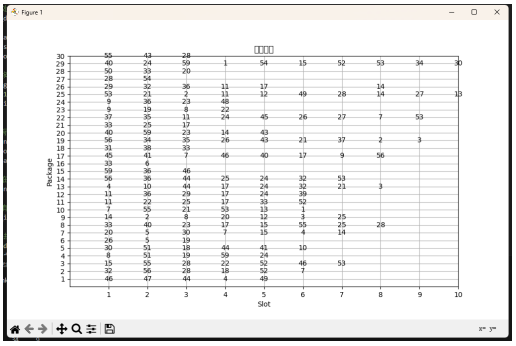

图 4-24: 表五数据结果图

从上图所示，经过模型求解后，表二的最小化墨盒总切换时间为 **258(min)**，表三的最小化墨盒总切换时间为 **442(min)**。

4.3 模型三

假设只有一台喷雾清洗机，并且不同颜色墨盒之间的切换时间不完全相同。每种包装对应的墨盒顺序是不同且固定的。在给定包装种类印刷顺序的情况下，建立总切换时间最小

化的数学模型，并根据附件 3 的数据计算总切换时间。

在问题二的基础上增加了一个限定条件，即每种包装对应的墨盒顺序是不同且固定的。由于插槽数量有限，无法一次性将所有的墨盒全部放置在插槽中。因此，在对一个种类的包装进行印刷时，放置在插槽中的墨盒集合需要包含该包装所需的墨盒。印刷滚筒和印压滚之间的间隙大小可调节，通过增大间隙，可使得空白印刷材料不与滚筒接触。

4.3.1 模型三建立

墨盒排列说明

根据附录说明，墨盒需要按照前后相对顺序放置。例如，柔性印刷机上有 4 个插槽，插槽序号分别为 1, 2, 3, 4，并且按照序号从小到大的顺序排列。某包装所需的墨盒编号分别为 5, 3, 8，并且这三个墨盒需要按照 5 在 3 之前，3 在 8 之前的顺序放置在插槽中。墨盒放置情况可以按照墨盒 5 放置在插槽 1 中，墨盒 3 放置在插槽 2 中，墨盒 8 放置在插槽 3 中的顺序放置；也可以按照墨盒 5 放置在插槽 1 中，墨盒 3 放置在插槽 2 中，墨盒 8 放置在插槽 4 中的顺序放置。

解决方案

为了确保墨盒按照前后相对顺序放置，我们需要在数学模型中增加一个墨盒相对位置约束。这个约束确保每个包装种类的墨盒按照给定的顺序排列在插槽中。通过添加例如 5, 3, 8 的约束条件，确保墨盒 5 在 3 之前，3 在 8 之前放置。

具体来说，我们建立一个整数规划模型，目标是最小化总切换时间。通过定义决策变量和辅助变量，并增加上述约束条件，确保模型能够满足问题的需求。以下是具体的数学模型构建和求解步骤：

目标函数

目标是最小化总切换时间:

$$\text{Min} \quad Z = \sum_{p \in P} \sum_{k \in S} \sum_{i \in C} \sum_{j \in C} T_{ij}^{k} x_{p,ij}^{k}$$

其中：

- Z 是总切换时间。
- P 是包装种类集合。
- S为插槽集合
- I 是墨盒的种类数量。
- t_{ij} 是从墨盒i 切换到墨盒j 的切换时间。
- $x_{p,ij}^{k}$ 是辅助变量，表示在第p 个包装种类到第$p+1$ 个包装种类的切换过程中，插槽k 从墨盒i 切换到墨盒j。

4.3.2 约束条件

每个包装种类的墨盒需求必须满足，并且顺序固定：

$$\sum_{k\in S} y_{i,p}^{k} = 1 \quad \forall p \in P, i \in M_p$$

其中 M_p 是包装种类 p 所需的墨盒集合。

墨盒的排列顺序不能变换：

$$\sum_{k'=1}^{k} z_{p,r_m}^{k'} \geq z_{p,r_{m+1}}^{k+1} \quad \forall p \in P, \forall m \in M_p, \forall k \in S$$

其中r_m 和r_{m+1} 分别表示包装种类p 所需的第m 和第m+1 个墨盒。 每个插槽只能容纳一个墨盒：

$$\sum_{i\in C} y_{i,k}^{p} \leq 1 \quad \forall p \in P, k \in S$$

辅助变量约束：

$$\begin{aligned}
&x_{p,i,j}^{k} \geq z_{p,i}^{k} + z_{p+1,j}^{k} - 1 \quad \forall k \in S, \forall i \in C, \forall j \in C, p \in P \\
&x_{p,i,j}^{k} \leq z_{p,i}^{k} \quad \forall k \in S, \forall i \in C, \forall j \in C, p \in P \qquad (14) \\
&x_{p,i,j}^{k} \leq z_{p+1,j}^{k} \quad \forall k \in S, \forall i \in C, \forall j \in C, p \in P \qquad (15)
\end{aligned}$$

如上，我们将问题三转化为了等价的 $0-1$ 整数线性规划模型：

$$\min \ Z = \sum_{k\in S}\sum_{i\in C}\sum_{j\in C} T_{ij}^{k} x_{ij}^{k},$$

$$s.t. \begin{cases}
\sum_{i\in C} y_i^k \leq 1, \quad \forall k \in S \\
\sum_{k'=1}^{k} z_{p,r_m}^{k'} \geq z_{p,r_{m+1}}^{k+1} \quad \forall p \in P, \forall m \in P, \forall k \in S \\
x_{p,ij}^{k} \geq z_{p,i}^{k} + z_{p+1,j}^{k} - 1 \quad \forall k \in S, \forall i \in C, \forall j \in C, p \in P \\
x_{p,ij}^{k} \leq z_{p,i}^{k} \quad \forall k \in S, \forall i \in C, \forall j \in C, p \in P \\
x_{p,ij}^{k} \leq z_{p+1,j}^{k} \quad \forall k \in S, \forall i \in C, \forall j \in C, p \in P
\end{cases}$$

4.3.3 模型三结果分析

根据所提出的算法计算得出，在问题三的约束条件下求解模型，可得固定墨盒顺序情况下墨盒总切换时间最短结果。

图 4-25: 表一数据结果图

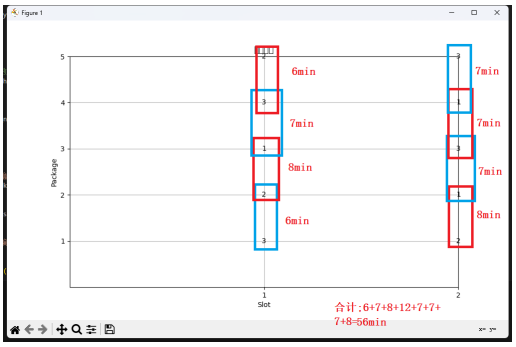

图 4-26: 表一切换时间计算示意

从上图所示，经过模型求解后，表一的最小化墨盒总切换时间为 **56(min)**。

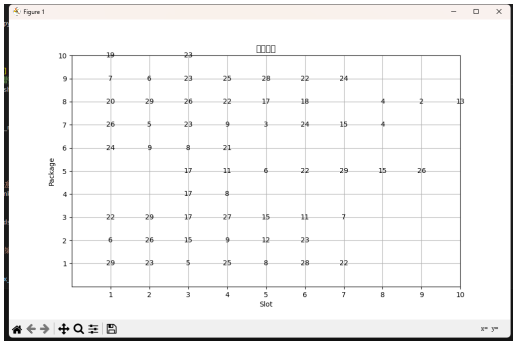

图 4-27: 表二数据结果图

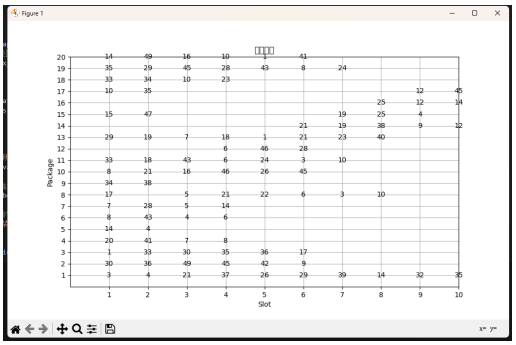

图 4-28: 表三数据结果图

从上图所示，经过模型求解后，表二的最小化墨盒总切换时间为 **189(min)**，表三的最小化墨盒总切换时间为 **239(min)**。

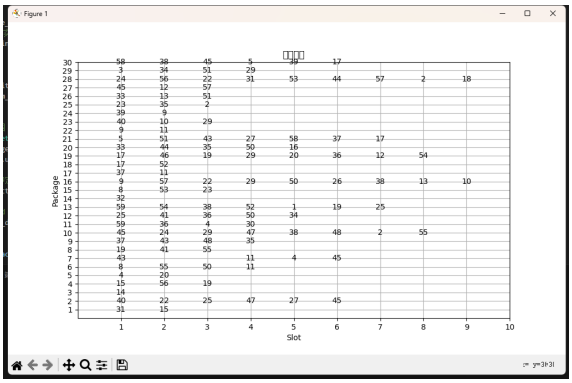

图 4-29: 表四数据结果图

从上图所示，经过模型求解后，表四的最小化墨盒总切换时间为 **288(min)**。

4.4 模型四

4.4.1 模型四建模

1. 变量定义：设变量 T_{ij}^k 表示在插槽 k 内墨盒从颜色 i 切换到颜色 j 的切换时间；设变量 y_i 表示插槽 k 上当前放置的墨盒编号；设 S 表示插槽集合，C 为不同颜色墨盒集合，x_{ij}^k 的定义与问题二相同，是一个表示插槽 k 中墨盒颜色是否由颜色 i 切换到颜色 j 的二进制变量。设 $z_{p,i,j}^k$ 是辅助二进制变量，表示在第 p 个包装种类到第 $p+1$ 个包装种类的切换过程中插槽 k 从墨盒 i 是否切换到墨盒 j。

2. 目标函数：与问题二、三相同，问题四的目标仍然是最小化总切换时间。将总切换时间表示为所有插槽上墨盒的切换时间总和：

$$\min \ \ Z = \sum_{k\in S}\sum_{i\in C}\sum_{j\in C} T_{ij}^k x_{ij}^k \tag{16}$$

3. 约束条件：

- 显然，在每个插槽中最多只能放置有一种颜色的墨盒，即该插槽有放置墨盒或并未使用。该约束条件表示如下：

$$\sum_{i\in C} y_i^k \le 1, \quad \forall k \in S \tag{17}$$

  当且仅当插槽 k 未放置有墨盒时上式左端为 0。

- 每种包装对应正确的颜色集合，即

$$y_i^k \ge \sum_{j\in C} x_{ji}^k, \quad \forall k \in S, \forall i \in C \tag{18}$$

- 插槽 k 中颜色为 i 的墨盒是否切换到颜色为 j 的墨盒应该与颜色为 i 或 j 的墨盒是否在插槽 k 中一致，即

$$y_i^k - y_j^k \ge x_{ij}^k \quad \forall k \in S, \forall i \in C, \forall j \in C \tag{19}$$

- 当前包装印刷时放置有墨盒的插槽个数一定大于等于印刷上个包装时放置有墨盒的插槽个数。即

$$\sum_{k\in S}\sum_{i\in C} y_{i,t+1}^k \ge \sum_{k\in S}\sum_{i\in C} y_{i,t}^k \tag{20}$$

- 每种包装种类印刷所需的墨盒顺序不同且固定。即

$$\sum_{k'=1}^{k} z_{p,r_m}^{k'} \ge z_{p,r_{m+1}}^{k+1} \quad \forall p, \forall m, \forall k \in \{1,2,\ldots,S-1\} \tag{21}$$

  其中 r_m 和 r_{m+1} 分别表示包装种类 p 所需的第 m 和第 $m+1$ 个墨盒。

- 辅助变量约束：

$$\begin{aligned} &x_{p,i,j}^k \geq z_{p,i}^k + z_{p+1,j}^k - 1 \quad \forall p,k,i,j \\ &x_{p,i,j}^k \leq z_{p,i}^k \quad \forall p,k,i,j \\ &x_{p,i,j}^k \leq z_{p+1,j}^k \quad \forall p,k,i,j \end{aligned} \tag{22}$$

如上，我们将问题四转化为等价的整数规划模型：

$$\min \ Z = \sum_{k\in S}\sum_{i\in C}\sum_{j\in C} T_{ij}^k x_{ij}^k,$$

$$s.t. \begin{cases} \sum_{i\in C} y_i^k \leq 1, \quad \forall k \in S \\ y_i^k \geq \sum_{j\in C} x_{ji}^k, \quad \forall k \in S, \forall i \in C \\ y_i^k - y_j^k \geq x_{ij}^k, \quad \forall k \in S, \forall i \in C, \forall j \in C \\ \sum_{k\in S}\sum_{i\in C} y_{i,t+1}^k \geq \sum_{k\in S}\sum_{i\in C} y_{i,t}^k \quad \forall k \in S, \forall i \in C, \forall j \in C \\ \sum_{k'=1}^{k} z_{p,r_m}^{k'} \geq z_{p,r_{m+1}}^{k+1} \quad \forall p, \forall m, \forall k \in S \\ x_{p,i,j}^k \geq z_{p,i}^k + z_{p+1,j}^k - 1 \quad \forall k \in S, \forall i \in C, \forall j \in C, p \in P \\ x_{p,i,j}^k \leq z_{p,i}^k \quad \forall k \in S, \forall i \in C, \forall j \in C, p \in P \\ x_{p,i,j}^k \leq z_{p+1,j}^k \quad \forall k \in S, \forall i \in C, \forall j \in C, p \in P \end{cases}$$

4.4.2 模型四求解

为了在印刷之前包装种类印刷顺序需要确定的情况下, 使总切换时间最小，本文提出了一种**贪心-模拟退火算法**, 首先通过建立动态规划模型，利用贪心算法计算在约束条件下的局部最优解，因贪心算法容易进入局部最优解而无法找到全局最优解，于是再利用退火算法使其以一定的概率来接受一个比当前解要差的解，因此有可能会跳出这个局部的最优解，达到全局的最优解。

在解决包装种类排序问题时，我们面临一个复杂的优化问题：不同颜色墨盒之间的切换时间不完全相同，每种包装对应的墨盒顺序固定，并且印刷顺序需要在印刷前确定。这种问题的解空间非常庞大，传统的优化方法如穷举法在计算量上不可行。而模拟退火算法提供了一种有效的启发式方法，能够在可接受的时间内找到近似最优解。

模拟退火算法的核心思想借鉴了物理中的退火过程，通过模拟固体加热和缓慢冷却的过程来寻找全局最优解。在高温阶段，系统具有较高的能量，能够跳出局部最优解，探索更大的解空间；随着温度的降低，系统逐渐稳定下来，最终收敛到全局最优或近似最优解。这一机制使得模拟退火算法特别适合解决包含多个局部最优解的复杂优化问题。

在我们的包装种类排序问题中，初始解可以通过简单的贪心算法快速生成，而模拟退火算法在此基础上进行进一步优化。通过随机交换包装种类的顺序，并根据目标函数值的变化决定是否接受新解，模拟退火算法能够有效避免陷入局部最优解。此外，逐步降低的温度使得算法在后期更倾向于接受较优解，确保最终解的质量。因此，模拟退火算法不仅能够处理大规模复杂问题，还能提供高质量的优化结果，是解决包装种类排序问题的理想选择。

(1) 模拟退火算法：

模拟退火算法（Simulated Annealing, SA）[5, 6] 是一种用于求解全局优化问题的随机化算法，其灵感来源于固体退火过程。该算法通过模拟物理退火过程中的加热和冷却过程，避免陷入局部最优，逐渐逼近全局最优解[7, 8]。以下是模拟退火算法的数学逻辑和步骤：**步骤一：**输入初始矩阵 X_0，参数设定初始温度 T_0，终止温度 T_r冷却速率v_c, 迭代记数$T_{0_{count}}$

步骤二：当前矩阵$X_w = X_0$, 当前温度$T_w = T_0$。

步骤三：进入循环

1、当此时温度 $T_w > T_r$, 对 X_w 进行部分更改，首先在 K 个包装种类中随机选定一个包装种类，该包装种类对于顺序的选择进行随机更改，随机从 1-K 的包装种类与原矩阵中第一次包装的顺序替换，其次在该包装给定墨盒中随机选定一个墨盒，更改该墨盒的返回次数上限，生成一个相邻解 $X_w + 1$∘ 2) 计算该矩阵 $X_w + 1$ 对应的墨盒切换总时间 $f_w + 1$。3) 判断 $\Delta f = f_{w+1} - f_w$。若 $\Delta f > 0$, 则接受为新解；若 $\Delta f < 0$, 则需要进行进一步的判断，对 $random(0, 1) - e^{\frac{\Delta f}{T_0}}$ 的大小进行判断，若大于 0, 则舍弃，并回到步骤 3 的 1); 若小于 0, 则也接受为新解。只要出现新解，$T_{0_{count}} + +$, 并判断 $T_{0_{count}}$ 是否大于设定迭代次数，若未超过，则进行下一步，否则，退出循环。4) 计算此时温度 $T_{w+1} = v_c * T_w$, 并判断此时温度是否小于终止温度，若大于，进行下一次循环；否则，退出循环。图 4 为问题一模拟退火的算法流程图：

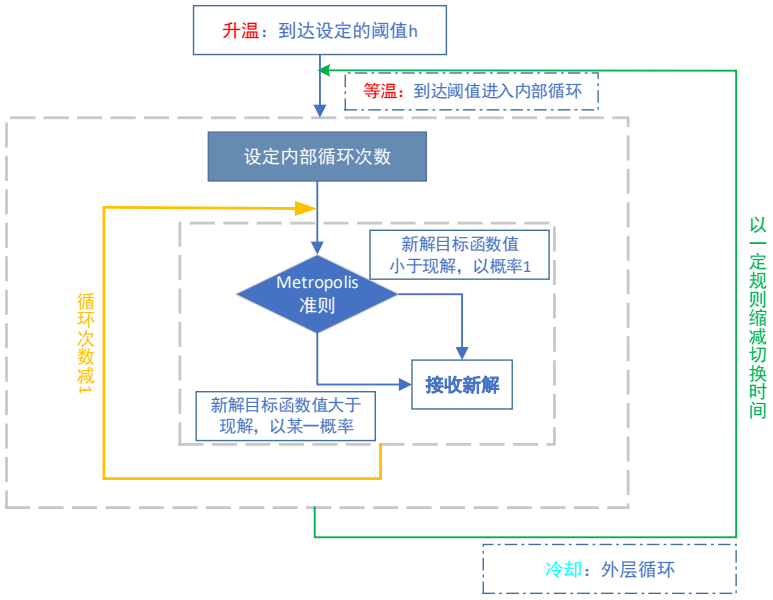

图 4-30：模拟退火流程图

4.4.3 模型四结果分析

包装种类顺序	插槽1	插槽2
3	4	2
5	7	3
1	7	5
2	2	5
4	8	
总计	21(min)	

图 4-31: 表一数据结果图

包装种类顺序	插槽1	插槽2	插槽3
1	7	8	
4	7	6	
3	3	6	4
5	1	6	
2	4	1	5
总计	37(min)		

图 4-32: 表二数据结果图

从上图所示，经过模型求解后，表一的最小化墨盒总切换时间为 **21(min)**，表二的最小化墨盒总切换时间为 **37(min)**。

包装种类顺序	插槽1	插槽2
7	3	1
5	7	5
3	6	
4	1	7
2	2	7
6	4	1
1	5	2
总计	71(min)	

图 4-33: 表三数据结果图

包装种类顺序	插槽1	插槽2	插槽3	插槽4	插槽5	插槽6	插槽7	插槽8	插槽9	插槽10
19	4	12	38	20	22	9	32	25	28	16
17	34	8	36	28	11	5	19			
6	18	27	3	1						
8	29	26	17	21	19	7	35	10		
3	38	20	37	4	17					
2	1	25	27	37	18					
14	28	37								
18	29	27								
1	23	27								
9	36	35	2							
12	20	8	5	37	3					
20	23	19	15	36	38	1	20	37	3	
5	17	7	16	25						
7	17	7	20	27	15					
10	37	15	18							
15	27	21								
16	32	39	18	34	33					
4	15	6	7	21	2	16	30	36	37	33
11	33	29								
13	13	20	12	38	9	37	32			
总计				332（min）						

图 4-34: 表四数据结果图

从上图所示，经过模型求解后，表三的最小化墨盒总切换时间为 **71(min)**，表四的最小化墨盒总切换时间为 **332(min)**。

4.5 模型五

4.5.1 模型五建立

1. **变量定义：**设 P 为包装种类集合；设所有包装印刷所需所有墨盒的集合 C，设 S 为印刷机的所有插槽集合；设 T_{ij} 为墨盒 i 切换到墨盒 j 所需要的时间；设 O_p 为第 p 个包

装印刷所需墨盒的顺序集合；设 y_p 表示第 p 个包装在整个印刷序列中的顺序；设 $z^k_{p,i,j}$ 是辅助二进制变量，表示在第 p 个包装种类到第 $p+1$ 个包装种类的切换过程中插槽 k 从墨盒 i 是否切换到墨盒 j；设 r_m 和 r_{m+1} 分别表示包装种类 p 所需的第 m 和第 $m+1$ 个墨盒；设 y^k_i 为二进制变量，表示墨盒 i 是否在插槽 k 中；设 $x^k_{ij,1}$ 为二进制变量，表示插槽 k 中是否由墨盒 i 切换到墨盒 j（由清洗机 1 清洗）；$x^k_{ij,2}$ 为二进制变量，表示插槽 k 中是否由墨盒 i 切换到墨盒 j（由清洗机 2 清洗）。

2. **目标函数**：由于有两台喷雾清洗机对待切换的插槽进行清洗，且必须在所有墨盒切换完毕后才能开始印刷，因此需要将两台清洗机的清洗次数乘以切换时间累加，所得最小总切换时间目标函数为

$$\min \ \ Z = \sum_{k\in S}\sum_{i\in C}\sum_{j\in C} T_{ij}(x^k_{ij,1} + x^k_{ij,2}) \tag{23}$$

3. **约束条件**：由题目分析可知，问题五由问题四的一台喷雾清洗机增加为两台，因此需要增加切换变量约束以表示清洗机 1 与清洗机 2 同时清洗不同待切换的插槽且不能同时清洗同一个插槽。此外，需要约束包装在整个印刷序列中的顺序。其余约束均类似。

   - 切换变量约束：

$$x^k_{ij,1} + x^k_{ij,2} \geq y^k_i + y^k_j - 1 \quad \forall k \in S, i \in C, j \in C \tag{24}$$

   - 包装顺序约束：

$$y_p \in \{1, \ldots, |P|\} \quad \forall p \in P \tag{25}$$

   我们将问题五转化为如下等价数学模型：

$$\min \ \ Z = \sum_{k\in S}\sum_{i\in C}\sum_{j\in C} T_{ij}(x^k_{ij,1} + x^k_{ij,2})$$

$$
s.t.\begin{cases}
\sum_{i\in C} y_i^k \le 1, \quad \forall k \in S \\
y_i^k \ge \sum_{j\in C} x_{ji,1}^k, \quad \forall k \in S, \forall i \in C \\
y_i^k \ge \sum_{j\in C} x_{ji,2}^k, \quad \forall k \in S, \forall i \in C \\
y_i^k - y_j^k \ge x_{ij,1}^k, \quad \forall k \in S, \forall i \in C, \forall j \in C \\
y_i^k - y_j^k \ge x_{ij,2}^k, \quad \forall k \in S, \forall i \in C, \forall j \in C \\
\sum_{k\in S}\sum_{i\in C} y_{i,t+1}^k \ge \sum_{k\in S}\sum_{i\in C} y_{i,t}^k \quad \forall k \in S, \forall i \in C, \forall j \in C \\
\sum_{k'=1}^{k} z_{p,r_m}^{k'} \ge z_{p,r_{m+1}}^{k+1} \quad \forall p, \forall m, \forall k \in S \\
x_{p,ij}^k \ge z_{p,i}^k + z_{p+1,j}^k - 1 \quad \forall k \in S, \forall i \in C, \forall j \in C, p \in P \\
x_{p,ij}^k \le z_{p,i}^k \quad \forall k \in S, \forall i \in C, \forall j \in C, p \in P \\
x_{p,ij}^k \le z_{p+1,j}^k \quad \forall k \in S, \forall i \in C, \forall j \in C, p \in P \\
x_{ij,1}^k + x_{ij,2}^k \ge y_i^k + y_j^k - 1 \quad \forall k \in S, i \in C, j \in C \\
y_p \in \{1, \ldots, |P|\} \quad \forall p \in P
\end{cases}
$$

4.5.2 模型五结果分析

包装种类顺序	插槽1	插槽2
1	4	1
2	2	1
5	2	4
4	3	2
3	4	3
总计	16(min)	

图 4-35: 表一数据结果图

包装种类顺序	插槽1	插槽2	插槽3	插槽4	插槽5	插槽6	插槽7	插槽8	插槽9	插槽10	插槽11	插槽12	插槽13	插槽14	插槽15
6	5	42	25												
1	11	7													
3	11	16	1	27	15	10	30	8	19	14	23	43	15	4	
4	23	19	30	48	12	10	44	31							
2	31	6	40	21	10	17	35	42	23	24	33				
10	2	24	38	6	39	9	4								
9	26	8	20	41	6	5	18	3	4	39	47	12	1	34	28
8	4	40	16	36	33	19	18	23	26	6	15	27	49	7	
5	4	38	41	10	26	6									
7	25	27	13	31	30	22	43	28	16	48	7	47	17		
总计	168(min)														

图 4-36: 表二数据结果图

从上图所示，经过模型求解后，表一的最小化墨盒总切换时间为 **16(min)**，表二的最小化墨盒总切换时间为 **168(min)**。

包装种类顺序	插槽1	插槽2	插槽3	插槽4	插槽5	插槽6	插槽7	插槽8	插槽9	插槽10
13	3	21	11	9						
1	19	49	22	41	21	39				
18	38	6	34							
12	21	13	1	41						
19	42	48	20	38	2	23	40	4	22	37
3	15	18	49	12	20	35	29	25		
2	28	8	22	4	48	7	19	41	9	33
11	5									
17	20	40	10	49	45	46	1	31		
5	21	49	23	26	14					
20	28	49	25	13						
4	35	6	46	4	28					
7	43	30								
15	16	12	41	15	4	31	28			
6	20	37	38	33	11	22	10	35	25	
10	40	39	4	3	31	22	27	35		
16	31	21	9	2	35	24	4	32	1	
9	6	27	34	48	36	26	12	29	20	37
14	6	43	41	23	35	17	46			
8	23	38	15	31						
总计					219(min)					

图 4-37: 表三数据结果图

包装种类顺序	插槽1	插槽2	插槽3	插槽4	插槽5	插槽6	插槽7	插槽8	插槽9	插槽10
20	46	56	20	2	21					
4	12	50	9	43	21	33	53	40	15	31
22	34	17	59							
30	43	17	59							
15	53	44								
1	23	49	12	5	32	43	51			
13	23	55	52	50	1	7	27			
24	43	17	19	24						
12	36	16	25	51	13	26				
21	36	40	17	56	10	25	24			
12	18	34	57	23	7	56	6			
14	59	12	14							
29	54									
3	38	39								
18	38	59	34	2	3	19	52	55	10	40
2	38	59	34	2	3	19	52	55	10	5
19	25	6								
17	25	26	33							
6	55	11	20	48	7					
23	29	23	35	12	44	20	17			
5	7	13	45	19	48	40	31	36	57	
7	18	5	25	13	38	4				
16	16	24	10							
8	14	22	49	9	13					
9	27	58								
26	47	35	34	49						
11	41	7	2	5	42	50	21			
10	27	21	52	14	11	6	1	31	21	27
25	17	21	32	14	10	6	30	31	24	27
27	49	26								
28	55	51	51							
总计					239min					

图 4-38: 表四数据结果图

从上图所示，经过模型求解后，表三的最小化墨盒总切换时间为 **219(min)**，表四的最小化墨盒总切换时间为 **239(min)**。

五　模型的评价与改进

5.1　模型评价

贪心算法是一种重要的算法设计策略。它基于一种贪婪的策略，每一步都做出在当前看来最好的选择，希望这样的局部最优解能够导向全局最优解[9]。尽管贪心算法并不总是能找到全局最优解，但在许多情况下，它能够提供相当接近最优解的有效解决方案。贪心算法的核心思想是在每一步都尽可能地获取最大或最小的好处，不考虑是否会影响未来的结果，只希望每一步都能做到最好。它是一种启发式算法，通常不能保证找到全局最优解，但可以找到一个接近最优解的解。

优点：简单易懂：贪心算法的实现相对简单，易于理解。高效：在许多情况下，贪心算法能够快速地找到解。近似最优解：贪心算法通常能够找到一个近似最优解。

整数规划模型在优化和决策领域具有重要地位，广泛应用于生产调度、物流管理、资源分配等实际问题中。其优势在于能够准确描述和解决涉及离散变量的复杂问题，提供精确的最优解。然而，整数规划问题通常具有较高的求解复杂度，尤其是大规模问题，需要依赖先进的求解器和算法。尽管如此，随着计算技术的进步，整数规划模型的应用效率和解决能力不断提升，使其成为解决实际优化问题的强大工具。

本文研究提出的模型通过结合整数规划、贪心算法和模拟退火算法，有效地解决了柔性印刷机墨盒切换问题，展示了其在优化计算上的实用性和高效性。**整数规划模型：**在给定包装印刷顺序的情况下，利用 PULP 库进行求解，成功最小化了墨盒切换次数和总切换时间，表现出色。**贪心算法：**在处理大规模问题时，通过分解问题和逐步求解局部最优，显著

提高了计算效率。尽管贪心算法在某些复杂约束条件下可能存在局限性，但它能够快速生成高质量的初始解，为进一步优化打下基础。**模拟退火算法：**通过随机扰动和概率接受机制，有效避免了陷入局部最优的问题，进一步优化了墨盒切换时间，特别是在需要确定包装种类印刷顺序的情况下，提升了全局优化能力。

5.2 模型改进与本文创新

在问题一中，我们在给定包装印刷顺序的情况下，仅需考虑每种包装对应的墨盒顺序。通过建立整数规划模型，并引入约束条件，使用 PULP 库进行直接求解，得到了优化的结果。

基于问题一，我们尝试在问题二中继续使用 PULP 库，通过分支定界法对构造的最优函数进行求解。然而，我们发现对于附件二中的前两个例子（表 1 和表 2），效果较好，但当墨盒数量增大、时间切换矩阵的维数增加到 30×30 时，计算量急剧增大，难以准确求解最优解。因此，我们采用贪心算法，将整个墨盒排序策略问题分解成多个步骤进行，在每一步骤中优先选取当前的最优方案，直到所有步骤完成。每一步都不考虑后续步骤的影响，也不回溯改变先前的选择。虽然贪心算法在数据量较小时可能获得全局最优解，但在问题三中加入了墨盒相对顺序不变的约束，即前面墨盒的插槽序号一定小于后面的墨盒，增加了复杂性。

在问题二的求解中，我们为方便后续计算，且经过测试不会对实验结果产生不良影响，在数据预处理时将包装种类-插槽矩阵第一行填充至最大插槽数。如图 4-2所示，取问题一附件第三张表前四行数据为例，本章第三节求解结果时已经展示此预处理的优势。

在问题四中，由于需要在印刷之前确定包装种类的印刷顺序，贪心算法显然不够用。因此，我们引入模拟退火算法处理包装种类的排序问题。首先，通过贪心算法获得一个局部最优解（最小切换时间），然后利用随机扰动（如交换包装顺序）有概率跳出局部最优，不断比较找到全局最优解。

在问题五中，我们沿用先前的建模方式和求解方法，但对于两台喷雾清洗机的分配问题，制定了规则，将两种包装种类之间墨盒切换时间总和相加，寻找切换时间相加超过总和一半且超出部分最少的方案。这种方法高效解决了喷雾清洗机的分配问题，但仍可能出现局部最优解的情况。

综上所述，贪心—模拟退火算法在计算复杂度上较低，且能够接近最优解。贪心算法通过快速生成初始解，确保了计算的高效性和可行性，而模拟退火算法通过概率机制逐步优化解，在寻找全局最优解时有效避免陷入局部最优。结合这两种算法的优势，既保留了贪心算法的高效性，又利用模拟退火算法的全局搜索能力，使得算法在保证计算速度的同时，能够接近最优解。这种组合方法不仅适用于墨盒切换问题，还在其他优化问题中展现出较高的实用性和有效性，大大提升了计算效率和解决问题的能力。

参考文献

[1] 曾壹, 张琦, 陈峰. 基于约束规划方法的高速铁路调整优化模型 [J]. 铁道学报,2019,41(04):1-9.

[2] A. Lim and Zhou Xu, "Searching optimal resequencing and feature assignment on an automated assembly line," 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05), 2005, pp. 8 pp.-498, doi: 10.1109/ICTAI.2005.114.

[3] Yu F, Lu C, Zhou J, et al. Mathematical model and knowledge-based iterated greedy algorithm for distributed assembly hybrid flow shop scheduling problem with dual-resource constraints[J]. Expert Systems with Applications, 2024, 239: 122434.

[4] McCall J. Genetic algorithms for modelling and optimisation[J]. Journal of computational and Applied Mathematics, 2005, 184(1): 205-222.

[5] Akay B, Karaboga D, Gorkemli B, et al. A survey on the artificial bee colony algorithm variants for binary, integer and mixed integer programming problems[J]. Applied Soft Computing, 2021, 106: 107351.

[6] McCall J. Genetic algorithms for modelling and optimisation[J]. Journal of computational and Applied Mathematics, 2005, 184(1): 205-222.

[7] 谷春红. 一种基于模拟退火算法自动排课系统设计 [J]. 中国科技信息,2023,(17):65-68.

[8] 张钧, 贺可太. 求解三维装箱问题的混合遗传模拟退火算法 [J]. 计算机工程与应用,2019,55(14):32-39+47.

[9] 曹玖新, 闵绘宇, 徐顺, 等. 基于启发式和贪心策略的社交网络影响最大化算法 [J]. 东南大学学报 (自然科学版),2016,46(05):950-956.

附录

问题一 python 代码。

```
%%%问题一
import pandas as pd
import matplotlib.pyplot as plt
from pulp import LpMinimize, LpProblem, LpVariable, lpSum, LpBinary

# 加载数据
file_path = 'C:/Users/zyh/Desktop/比赛 代码/附件数据（B题）/附件1/Ins1_5_10_2.xlsx' # 修改为您的实际文件路径
sheet1 = pd.read_excel(file_path, sheet_name='包装-墨盒-插槽')
sheet2 = pd.read_excel(file_path, sheet_name='包装种类及其所需墨盒')

# 数据预处理
switching_data = sheet1.dropna().astype(int)
package_data = sheet2

num_packages = package_data.shape[0]
num_slots = 2
num_inkboxes = 10

# 检查实际的墨盒编号范围
actual_inkbox_ids = set()
for inkboxes in package_data['所需墨盒编号']:
actual_inkbox_ids.update(eval(inkboxes))

# 建立整数规划模型
model = LpProblem(name="minimize_switching", sense=LpMinimize)

# 定义决策变量
x = LpVariable.dicts("x", ((i, j, k) for i in range(1, num_packages + 1) for j in actual_inkbox_ids for k
    in range(1, num_slots + 1)), cat=LpBinary)

# 定义辅助变量，用于表示两个决策变量之间的切换
y = LpVariable.dicts("y", ((p, j, k) for p in range(1, num_packages) for j in actual_inkbox_ids for k in
    range(1, num_slots + 1)), cat=LpBinary)

# 定义目标函数
objective = lpSum(y[p, j, k] for p in range(1, num_packages) for j in actual_inkbox_ids for k in range(1,
    num_slots + 1))
model += objective

# 设置约束条件
# 每个包装种类的墨盒需求必须满足
for idx, row in package_data.iterrows():
```

```
    package_id = row['包装种类编号']
    required_inkboxes = eval(row['所需墨盒编号'])
    for inkbox in required_inkboxes:
    model += lpSum(x[package_id, inkbox, k] for k in range(1, num_slots + 1)) == 1

    # 每个插槽只能容纳一个墨盒
    for i in range(1, num_packages + 1):
    for k in range(1, num_slots + 1):
    model += lpSum(x[i, j, k] for j in actual_inkbox_ids) == 1

    # 辅助变量约束
    for p in range(1, num_packages):
    for j in actual_inkbox_ids:
    for k in range(1, num_slots + 1):
    model += y[p, j, k] >= x[p, j, k] - x[p + 1, j, k]
    model += y[p, j, k] >= x[p + 1, j, k] - x[p, j, k]

    # 求解模型
    model.solve()

    # 提取解
    solution = {(i, j, k): x[i, j, k].varValue for i in range(1, num_packages + 1) for j in actual_inkbox_ids
        for k in range(1, num_slots + 1)}

    # 构建结果 DataFrame
    result_data = []
    for key, value in solution.items():
    if value == 1:
    result_data.append([key[0], key[1], key[2]])

    result_df = pd.DataFrame(result_data, columns=['Package', 'Inkbox', 'Slot'])

    # 输出结果 DataFrame
    print("优化结果: ")
    print(result_df)

    # 计算并输出切换次数
    switch_count = 0
    switch_data = []
    for p in range(1, num_packages):
    for j in actual_inkbox_ids:
    for k in range(1, num_slots + 1):
    if y[p, j, k].varValue == 1:
    switch_count += 1
    switch_data.append([p, j, k])

```

```
print(f"\n总的切换次数: {switch_count}")

# 绘制优化结果
fig, ax = plt.subplots(figsize=(10, 6))
for idx, row in result_df.iterrows():
ax.text(row['Slot'], row['Package'], str(row['Inkbox']), ha='center', va='center')

ax.set_xticks(range(1, num_slots + 1))
ax.set_yticks(range(1, num_packages + 1))
ax.set_xticklabels(range(1, num_slots + 1))
ax.set_yticklabels(range(1, num_packages + 1))
ax.set_xlabel('Slot')
ax.set_ylabel('Package')
ax.set_title('优化结果')
ax.grid(True)
plt.show()

# 绘制切换次数
fig, ax = plt.subplots(figsize=(10, 6))
switch_df = pd.DataFrame(switch_data, columns=['FromPackage', 'Inkbox', 'Slot'])
for idx, row in switch_df.iterrows():
ax.arrow(row['Slot'], row['FromPackage'], 0, 1, head_width=0.2, head_length=0.2, fc='blue', ec='blue')

ax.set_xticks(range(1, num_slots + 1))
ax.set_yticks(range(1, num_packages + 1))
ax.set_xticklabels(range(1, num_slots + 1))
ax.set_yticklabels(range(1, num_packages + 1))
ax.set_xlabel('Slot')
ax.set_ylabel('Package')
ax.set_title('切换次数')
ax.grid(True)
plt.show()
```

问题二 python 代码。

```
%%%问题二 贪心算法
import pandas as pd
import numpy as np

# 加载数据
file_path = 'C:/Users/zyh/Desktop/比赛 代码/附件数据（B题）/附件2/Ins3_10_30_10.xlsx' # 修改为您的实际文件路
    径
sheet1 = pd.read_excel(file_path, sheet_name='墨盒切换时间', index_col=0)
sheet2 = pd.read_excel(file_path, sheet_name='包装种类及其所需墨盒')

# 打印读取的数据以确认
```

```
    print("Switching Time Matrix:")
    print(sheet1)
    print("Package Data:")
    print(sheet2)

    # 将切换时间矩阵转换为NumPy数组
    switching_matrix_np = sheet1.to_numpy()

    # 数据预处理
    package_data = sheet2

    num_packages = package_data.shape[0]
    num_slots = 10 # 根据实际情况设置插槽数量
    num_inkboxes = switching_matrix_np.shape[0] # 根据切换时间矩阵的大小自动设置

    # 定义切换时间矩阵
    switching_time = {
        i + 1: {j + 1: switching_matrix_np[i][j] for j in range(num_inkboxes)}
        for i in range(num_inkboxes)
    }

    # 检查实际的墨盒编号范围
    actual_inkbox_ids = set()
    for inkboxes in package_data['所需墨盒编号']:
    actual_inkbox_ids.update(eval(inkboxes))

    # 过滤出在切换时间矩阵中定义的墨盒编号
    actual_inkbox_ids = actual_inkbox_ids.intersection(switching_time.keys())

    # 确认所有的 package_id 都在范围内
    package_ids = package_data['包装种类编号'].tolist()

    # 贪心算法
    def greedy_schedule(packages, inkbox_order, switching_time):
    schedule = []
    total_switching_time = 0
    current_slots = [-1] * num_slots # 初始化插槽为空

    for package_id in packages:
    required_inkboxes = eval(package_data.loc[package_data['包装种类编号'] == package_id, '所需墨盒编号'].
        values[0])
    current_schedule = [-1] * num_slots
    used_slots = 0

    # 放置需要的墨盒
    for inkbox in required_inkboxes:
```

```
    for slot in range(num_slots):
    if current_slots[slot] == inkbox:
    current_schedule[slot] = inkbox
    break
    else:
    for slot in range(num_slots):
    if current_schedule[slot] == -1:
    current_schedule[slot] = inkbox
    used_slots += 1
    break

    # 计算切换时间
    for slot in range(num_slots):
    if current_slots[slot] != current_schedule[slot] and current_schedule[slot] != -1:
    if current_slots[slot] != -1:
    total_switching_time += switching_time[current_slots[slot]][current_schedule[slot]]
    else:
    total_switching_time += 0 # 新插入墨盒，不需要切换时间

    current_slots = current_schedule[:]
    schedule.append(current_schedule)

    return schedule, total_switching_time

    # 使用贪心算法计算优化结果
    schedule, total_switching_time = greedy_schedule(package_ids, actual_inkbox_ids, switching_time)

    # 构建结果 DataFrame
    result_data = []
    for p, current_schedule in enumerate(schedule):
    for slot, inkbox in enumerate(current_schedule):
    if inkbox != -1:
    result_data.append([package_ids[p], inkbox, slot + 1])

    result_df = pd.DataFrame(result_data, columns=['Package', 'Inkbox', 'Slot'])

    # 输出结果 DataFrame
    print("优化结果：")
    print(result_df)

    print(f"Total switching time: {total_switching_time}")

    # 绘制优化结果
    import matplotlib.pyplot as plt

    fig, ax = plt.subplots(figsize=(10, 6))
```

```
for idx, row in result_df.iterrows():
ax.text(row['Slot'], row['Package'], str(row['Inkbox']), ha='center', va='center')

ax.set_xticks(range(1, num_slots + 1))
ax.set_yticks(range(1, num_packages + 1))
ax.set_xticklabels(range(1, num_slots + 1))
ax.set_yticklabels(range(1, num_packages + 1))
ax.set_xlabel('Slot')
ax.set_ylabel('Package')
ax.set_title('优化结果')
ax.grid(True)
plt.show()

%%%问题二 整数规划
import pandas as pd
import numpy as np
from pulp import LpMinimize, LpProblem, LpVariable, lpSum, LpBinary

# 加载数据
file_path = 'C:/Users/zyh/Desktop/比赛 代码/附件数据（B题）/附件2/Ins5_30_60_10.xlsx' # 修改为您的实际文件路
    径
sheet1 = pd.read_excel(file_path, sheet_name='墨盒切换时间', index_col=0)
sheet2 = pd.read_excel(file_path, sheet_name='包装种类及其所需墨盒')

# 打印读取的数据以确认
print("Switching Time Matrix:")
print(sheet1)
print("Package Data:")
print(sheet2)

# 将切换时间矩阵转换为NumPy数组
switching_matrix_np = sheet1.to_numpy()

# 打印矩阵形状以确认
print("Switching Matrix Shape:", switching_matrix_np.shape)

# 数据预处理
package_data = sheet2

num_packages = package_data.shape[0]
num_inkboxes = switching_matrix_np.shape[0] # 根据切换时间矩阵的大小自动设置

# 定义切换时间矩阵
switching_time = {
    i + 1: {j + 1: switching_matrix_np[i][j] for j in range(num_inkboxes)}
    for i in range(num_inkboxes)
```

```
}

# 检查实际的墨盒编号范围
actual_inkbox_ids = set()
for inkboxes in package_data['所需墨盒编号']:
actual_inkbox_ids.update(eval(inkboxes))

# 打印 actual_inkbox_ids 以确认其范围
print(f"Actual inkbox IDs: {actual_inkbox_ids}")

# 过滤出在切换时间矩阵中定义的墨盒编号
actual_inkbox_ids = actual_inkbox_ids.intersection(switching_time.keys())

# 再次打印过滤后的 actual_inkbox_ids 以确认其范围
print(f"Filtered actual inkbox IDs: {actual_inkbox_ids}")

# 确认所有的 package_id 都在范围内
package_ids = package_data['包装种类编号'].tolist()
print(f"Package IDs: {package_ids}")

# 建立整数规划模型
model = LpProblem(name="minimize_switching_time", sense=LpMinimize)

# 定义决策变量
x = LpVariable.dicts("x", ((i, j) for i in package_ids for j in actual_inkbox_ids), cat=LpBinary)

# 定义辅助变量，用于表示两个决策变量之间的切换
y = LpVariable.dicts("y", ((p, i, j) for p in range(1, num_packages) for i in actual_inkbox_ids for j in
    actual_inkbox_ids), cat=LpBinary)

# 定义目标函数
objective = lpSum(switching_time[i][j] * y[p, i, j] for p in range(1, num_packages) for i in
    actual_inkbox_ids for j in actual_inkbox_ids)
model += objective

# 设置约束条件
# 每个包装种类的墨盒需求必须满足
for idx, row in package_data.iterrows():
package_id = row['包装种类编号']
required_inkboxes = eval(row['所需墨盒编号'])
for inkbox in required_inkboxes:
model += lpSum(x[package_id, inkbox]) == 1

# 每个插槽只能容纳一个墨盒
for i in package_ids:
model += lpSum(x[i, j] for j in actual_inkbox_ids) == len(eval(package_data.loc[package_data['包装种类编号'
```

```
        ] == i, '所需墨盒编号'].values[0]))

    # 辅助变量约束
    for p in range(1, num_packages):
    for i in actual_inkbox_ids:
    for j in actual_inkbox_ids:
    model += y[p, i, j] >= x[p, i] + x[p + 1, j] - 1
    model += y[p, i, j] <= x[p, i]
    model += y[p, i, j] <= x[p + 1, j]

    # 求解模型
    model.solve()

    # 提取解
    solution = {(i, j): x[i, j].varValue for i in package_ids for j in actual_inkbox_ids}

    # 输出结果
    for key, value in solution.items():
    if value == 1:
    print(f"Package {key[0]} uses inkbox {key[1]}")

    # 计算并输出总切换时间
    total_switching_time = 0
    for p in range(1, num_packages):
    for i in actual_inkbox_ids:
    for j in actual_inkbox_ids:
    if y[p, i, j].varValue == 1:
    total_switching_time += switching_time[i][j]

    print(f"Total switching time: {total_switching_time}")
```

问题三 python 代码。

```
    %%%问题三
import pandas as pd
import numpy as np

# 加载数据
file_path = 'C:/Users/zyh/Desktop/比赛 代码/附件数据（B题）/附件3/Ins4_30_60_10.xlsx' # 确保路径和文件名正确
sheet1 = pd.read_excel(file_path, sheet_name='墨盒切换时间', index_col=0)
sheet2 = pd.read_excel(file_path, sheet_name='包装种类及其所需墨盒')

# 打印读取的数据以确认
print("Switching Time Matrix:")
print(sheet1)
print("Package Data:")
```

```
print(sheet2)

# 将切换时间矩阵转换为NumPy数组
switching_matrix_np = sheet1.to_numpy()

# 数据预处理
package_data = sheet2

num_packages = package_data.shape[0]
num_slots = 10 # 根据实际情况设置插槽数量
num_inkboxes = switching_matrix_np.shape[0] # 根据切换时间矩阵的大小自动设置

# 定义切换时间矩阵
switching_time = {
    i + 1: {j + 1: switching_matrix_np[i][j] for j in range(num_inkboxes)}
    for i in range(num_inkboxes)
}

# 检查实际的墨盒编号范围
actual_inkbox_ids = set()
for inkboxes in package_data['所需墨盒编号']:
actual_inkbox_ids.update(eval(inkboxes))

# 过滤出在切换时间矩阵中定义的墨盒编号
actual_inkbox_ids = actual_inkbox_ids.intersection(switching_time.keys())

# 确认所有的 package_id 都在范围内
package_ids = package_data['包装种类编号'].tolist()

# 贪心算法
def greedy_schedule(packages, inkbox_order, switching_time):
schedule = []
total_switching_time = 0
current_slots = [-1] * num_slots # 初始化插槽为空

for package_id in packages:
required_inkboxes = eval(package_data.loc[package_data['包装种类编号'] == package_id, '所需墨盒编号'].values
    [0])
current_schedule = [-1] * num_slots
used_slots = 0

# 放置需要的墨盒并保证顺序
last_filled_slot = -1
for inkbox in required_inkboxes:
for slot in range(last_filled_slot + 1, num_slots):
if current_slots[slot] == inkbox:
```

```
current_schedule[slot] = inkbox
last_filled_slot = slot
break
else:
for slot in range(last_filled_slot + 1, num_slots):
if current_schedule[slot] == -1:
current_schedule[slot] = inkbox
last_filled_slot = slot
used_slots += 1
break

# 计算切换时间
for slot in range(num_slots):
if current_slots[slot] != current_schedule[slot] and current_schedule[slot] != -1:
if current_slots[slot] != -1:
total_switching_time += switching_time[current_slots[slot]][current_schedule[slot]]
else:
total_switching_time += 0 # 新插入墨盒，不需要切换时间

current_slots = current_schedule[:]
schedule.append(current_schedule)

return schedule, total_switching_time

# 使用贪心算法计算优化结果
schedule, total_switching_time = greedy_schedule(package_ids, actual_inkbox_ids, switching_time)

# 构建结果 DataFrame
result_data = []
for p, current_schedule in enumerate(schedule):
for slot, inkbox in enumerate(current_schedule):
if inkbox != -1:
result_data.append([package_ids[p], inkbox, slot + 1])

result_df = pd.DataFrame(result_data, columns=['Package', 'Inkbox', 'Slot'])

# 输出结果 DataFrame
print("优化结果：")
print(result_df)

print(f"Total switching time: {total_switching_time}")

# 绘制优化结果
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 6))
```

```python
for idx, row in result_df.iterrows():
ax.text(row['Slot'], row['Package'], str(row['Inkbox']), ha='center', va='center')

ax.set_xticks(range(1, num_slots + 1))
ax.set_yticks(range(1, num_packages + 1))
ax.set_xticklabels(range(1, num_slots + 1))
ax.set_yticklabels(range(1, num_packages + 1))
ax.set_xlabel('Slot')
ax.set_ylabel('Package')
ax.set_title('优化结果')
ax.grid(True)
plt.show()
```

问题四 python 代码。

```python
    %%%问题四
import pandas as pd
import numpy as np
import random

# 加载数据
file_path = 'C:/Users/zyh/Desktop/比赛 代码/附件数据（B题）/附件4/Ins4_20_40_10.xlsx' # 确保路径和文件名正确
sheet1 = pd.read_excel(file_path, sheet_name='墨盒切换时间', index_col=0)
sheet2 = pd.read_excel(file_path, sheet_name='包装种类及其所需墨盒')

# 打印读取的数据以确认
print("Switching Time Matrix:")
print(sheet1)
print("Package Data:")
print(sheet2)

# 将切换时间矩阵转换为NumPy数组
switching_matrix_np = sheet1.to_numpy()

# 数据预处理
package_data = sheet2

num_packages = package_data.shape[0]
num_slots =10 # 根据实际情况设置插槽数量
num_inkboxes = switching_matrix_np.shape[0] # 根据切换时间矩阵的大小自动设置

# 定义切换时间矩阵
switching_time = {
    i + 1: {j + 1: switching_matrix_np[i][j] for j in range(num_inkboxes)}
    for i in range(num_inkboxes)
}
```

```
32
33  # 检查实际的墨盒编号范围
34  actual_inkbox_ids = set()
35  for inkboxes in package_data['所需墨盒编号']:
36  actual_inkbox_ids.update(eval(inkboxes))
37
38  # 过滤出在切换时间矩阵中定义的墨盒编号
39  actual_inkbox_ids = actual_inkbox_ids.intersection(switching_time.keys())
40
41  # 确认所有的 package_id 都在范围内
42  package_ids = package_data['包装种类编号'].tolist()
43
44  # 改进后的贪心算法
45  def greedy_schedule_with_fixed_order(packages, inkbox_order, switching_time, num_slots):
46  schedule = []
47  total_switching_time = 0
48  current_slots = [-1] * num_slots # 初始化插槽为空
49  for package_id in packages:
50  required_inkboxes = eval(package_data.loc[package_data['包装种类编号'] == package_id, '所需墨盒编号'].values
        [0])
51  current_schedule = [-1] * num_slots
52  inkbox_idx = 0
53
54  # 放置需要的墨盒并保证顺序
55  for slot in range(num_slots):
56  if inkbox_idx < len(required_inkboxes):
57  inkbox = required_inkboxes[inkbox_idx]
58  if inkbox not in current_schedule: # 避免重复放置同一个墨盒
59  current_schedule[slot] = inkbox
60  inkbox_idx += 1
61
62  # 计算切换时间
63  for slot in range(num_slots):
64  if current_slots[slot] != current_schedule[slot] and current_slots[slot] != -1 and current_schedule[slot] !=
        -1:
65  total_switching_time += switching_time[current_slots[slot]][current_schedule[slot]]
66
67  # 更新当前插槽配置
68  for slot in range(num_slots):
69  if current_slots[slot] != current_schedule[slot] and current_schedule[slot] != -1:
70  current_slots[slot] = current_schedule[slot]
71
72  # 输出当前插槽状态
73  print(f"After processing package {package_id}, slots state: {current_slots}")
74
75  schedule.append(current_schedule)
```

```
76
77  return schedule, total_switching_time
78
79  # 模拟退火算法
80  def simulated_annealing(packages, inkbox_order, switching_time, num_slots, initial_temp, cooling_rate,
        max_iter):
81  current_order = packages[:]
82  current_schedule, current_cost = greedy_schedule_with_fixed_order(current_order, inkbox_order, switching_time
        , num_slots)
83  best_order = current_order[:]
84  best_cost = current_cost
85
86  temp = initial_temp
87
88  for i in range(max_iter):
89  # 生成一个新的解
90  new_order = current_order[:]
91  idx1, idx2 = random.sample(range(len(new_order)), 2)
92  new_order[idx1], new_order[idx2] = new_order[idx2], new_order[idx1]
93
94  new_schedule, new_cost = greedy_schedule_with_fixed_order(new_order, inkbox_order, switching_time, num_slots)
95
96  # 接受新的解的概率
97  if new_cost < current_cost or random.uniform(0, 1) < np.exp((current_cost - new_cost) / temp):
98  current_order = new_order
99  current_schedule = new_schedule
100 current_cost = new_cost
101
102 if new_cost < best_cost:
103 best_order = new_order
104 best_cost = new_cost
105
106 temp *= cooling_rate
107
108 return best_order, best_cost
109
110 # 设置模拟退火算法的参数
111 initial_temp = 10000
112 cooling_rate = 0.99 # 增加冷却率
113 max_iter = 5000 # 增加迭代次数
114
115 # 使用模拟退火算法优化印刷顺序
116 best_order, best_cost = simulated_annealing(package_ids, actual_inkbox_ids, switching_time, num_slots,
        initial_temp, cooling_rate, max_iter)
117
118 # 使用贪心算法计算最佳顺序的优化结果
```

```
schedule, total_switching_time = greedy_schedule_with_fixed_order(best_order, actual_inkbox_ids,
    switching_time, num_slots)

# 构建结果 DataFrame
result_data = []
for p, current_schedule in enumerate(schedule):
slot_info = {f"Slot {slot + 1}": inkbox for slot, inkbox in enumerate(current_schedule) if inkbox != -1}
result_data.append([best_order[p], slot_info])

result_df = pd.DataFrame(result_data, columns=['包装种类编号', '插槽信息'])

# 输出结果 DataFrame
pd.set_option('display.width', 300) # 设置字符显示宽度 pd.set_option('display.max_rows', None) # 设置显示最大行
pd.set_option('display.max_columns', None) # 设置显示最大列，None为显示所有列
print("优化结果: ")
print(result_df)

print(f"Total switching time: {total_switching_time}")

# 绘制优化结果
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 6))
for idx, row in result_df.iterrows():
inkbox_info = ', '.join([f"{slot}: {inkbox}" for slot, inkbox in row['插槽信息'].items()])
ax.text(len(row['插槽信息']), row['包装种类编号'], inkbox_info, ha='center', va='center')

ax.set_xticks(range(1, num_slots + 1))
ax.set_yticks(range(1, num_packages + 1))
ax.set_xticklabels(range(1, num_slots + 1))
ax.set_yticklabels(range(1, num_packages + 1))
ax.set_xlabel('Slot')
ax.set_ylabel('Package')
ax.set_title('优化结果')
ax.grid(True)
plt.show()
```

问题五 python 代码。

```
    %%%问题五
import pandas as pd
import numpy as np
import random

# 加载数据
file_path = 'C:/Users/zyh/Desktop/比赛 代码/附件数据（B题）/附件5/Ins4_30_60_10.xlsx' # 确保路径和文件名正确
```

```
sheet1 = pd.read_excel(file_path, sheet_name='墨盒切换时间', index_col=0)
sheet2 = pd.read_excel(file_path, sheet_name='包装种类及其所需墨盒')

# 打印读取的数据以确认
print("Switching Time Matrix:")
print(sheet1)
print("Package Data:")
print(sheet2)

# 将切换时间矩阵转换为NumPy数组
switching_matrix_np = sheet1.to_numpy()

# 数据预处理
package_data = sheet2

num_packages = package_data.shape[0]
num_slots = 10 # 根据实际情况设置插槽数量
num_inkboxes = switching_matrix_np.shape[0] # 根据切换时间矩阵的大小自动设置

# 定义切换时间矩阵
switching_time = {
    i + 1: {j + 1: switching_matrix_np[i][j] for j in range(num_inkboxes)}
    for i in range(num_inkboxes)
}

# 检查实际的墨盒编号范围
actual_inkbox_ids = set()
for inkboxes in package_data['所需墨盒编号']:
actual_inkbox_ids.update(eval(inkboxes))

# 过滤出在切换时间矩阵中定义的墨盒编号
actual_inkbox_ids = actual_inkbox_ids.intersection(switching_time.keys())

# 确认所有的 package_id 都在范围内
package_ids = package_data['包装种类编号'].tolist()

# 改进后的贪心算法
def greedy_schedule_with_two_machines(packages, inkbox_order, switching_time, num_slots):
schedule = []
total_switching_time = 0

# 初始化插槽
machine1_slots = [-1] * (num_slots // 2 + num_slots % 2) # 初始化插槽为空
machine2_slots = [-1] * (num_slots // 2) # 初始化插槽为空

for package_id in packages:
```

```
required_inkboxes = eval(package_data.loc[package_data['包装种类编号'] == package_id, '所需墨盒编号'].values[0])
current_schedule1 = [-1] * (num_slots // 2 + num_slots % 2)
current_schedule2 = [-1] * (num_slots // 2)
inkbox_idx = 0

# 放置需要的墨盒并保证顺序
for slot in range(num_slots):
if inkbox_idx < len(required_inkboxes):
inkbox = required_inkboxes[inkbox_idx]
if slot % 2 == 0: # 偶数插槽分配给machine1
if inkbox not in current_schedule1: # 避免重复放置同一个墨盒
current_schedule1[slot // 2] = inkbox
inkbox_idx += 1
else: # 奇数插槽分配给machine2
if inkbox not in current_schedule2: # 避免重复放置同一个墨盒
current_schedule2[slot // 2] = inkbox
inkbox_idx += 1

# 计算切换时间
machine1_time = 0
machine2_time = 0
for slot in range(len(machine1_slots)):
if machine1_slots[slot] != current_schedule1[slot] and machine1_slots[slot] != -1 and current_schedule1[slot] != -1:
machine1_time += switching_time[machine1_slots[slot]][current_schedule1[slot]]
for slot in range(len(machine2_slots)):
if machine2_slots[slot] != current_schedule2[slot] and machine2_slots[slot] != -1 and current_schedule2[slot] != -1:
machine2_time += switching_time[machine2_slots[slot]][current_schedule2[slot]]

max_time = max(machine1_time, machine2_time)
total_switching_time += max_time

# 更新两个插槽配置
for slot in range(len(machine1_slots)):
if machine1_slots[slot] != current_schedule1[slot] and current_schedule1[slot] != -1:
machine1_slots[slot] = current_schedule1[slot]
for slot in range(len(machine2_slots)):
if machine2_slots[slot] != current_schedule2[slot] and current_schedule2[slot] != -1:
machine2_slots[slot] = current_schedule2[slot]

# 输出当前插槽状态
print(f"After processing package {package_id}, machine1 slots state: {machine1_slots}, machine2 slots state: {machine2_slots}")

```

```
schedule.append((current_schedule1, current_schedule2))

return schedule, total_switching_time

# 模拟退火算法
def simulated_annealing(packages, inkbox_order, switching_time, num_slots, initial_temp, cooling_rate,
    max_iter):
current_order = packages[:]
current_schedule, current_cost = greedy_schedule_with_two_machines(current_order, inkbox_order,
    switching_time, num_slots)
best_order = current_order[:]
best_cost = current_cost

temp = initial_temp

for i in range(max_iter):
# 生成一个新的解
new_order = current_order[:]
idx1, idx2 = random.sample(range(len(new_order)), 2)
new_order[idx1], new_order[idx2] = new_order[idx2], new_order[idx1]

new_schedule, new_cost = greedy_schedule_with_two_machines(new_order, inkbox_order, switching_time, num_slots
    )

# 接受新的解的概率
if new_cost < current_cost or random.uniform(0, 1) < np.exp((current_cost - new_cost) / temp):
current_order = new_order
current_schedule = new_schedule
current_cost = new_cost

if new_cost < best_cost:
best_order = new_order
best_cost = new_cost

temp *= cooling_rate

return best_order, best_cost

# 设置模拟退火算法的参数
initial_temp = 10000
cooling_rate = 0.99 # 增加冷却率
max_iter = 5000 # 增加迭代次数

# 使用模拟退火算法优化印刷顺序
best_order, best_cost = simulated_annealing(package_ids, actual_inkbox_ids, switching_time, num_slots,
    initial_temp, cooling_rate, max_iter)
```

```
# 使用贪心算法计算最佳顺序的优化结果
schedule, total_switching_time = greedy_schedule_with_two_machines(best_order, actual_inkbox_ids,
    switching_time, num_slots)
# 构建结果 DataFrame
result_data = []
for p, (current_schedule1, current_schedule2) in enumerate(schedule):
slot_info1 = {f"Machine 1 Slot {slot + 1}": inkbox for slot, inkbox in enumerate(current_schedule1) if inkbox
     != -1}
slot_info2 = {f"Machine 2 Slot {slot + 1}": inkbox for slot, inkbox in enumerate(current_schedule2) if inkbox
     != -1}
slot_info = {**slot_info1, **slot_info2}
result_data.append([best_order[p], slot_info])

result_df = pd.DataFrame(result_data, columns=['包装种类编号', '插槽信息'])

# 输出结果 DataFrame
pd.set_option('display.width', 300) # 设置字符显示宽度
pd.set_option('display.max_rows', None) # 设置显示最大行
pd.set_option('display.max_columns', None) # 设置显示最大列，None为显示所有列
print("优化结果: ")
print(result_df)

print(f"Total switching time: {total_switching_time}")

# 将结果保存到 Excel 文件
output_file_path = 'C:/Users/zyh/Desktop/比赛 代码/附件数据（B题）/优化结果_两台清洗机4.xlsx'
result_df.to_excel(output_file_path, index=False)

# 绘制优化结果
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 6))
for idx, row in result_df.iterrows():
inkbox_info = ', '.join([f"{slot}: {inkbox}" for slot, inkbox in row['插槽信息'].items()])
ax.text(len(row['插槽信息']), row['包装种类编号'], inkbox_info, ha='center', va='center')

ax.set_xticks(range(1, num_slots + 1))
ax.set_yticks(range(1, num_packages + 1))
ax.set_xticklabels(range(1, num_slots + 1))
ax.set_yticklabels(range(1, num_packages + 1))
ax.set_xlabel('Slot')
ax.set_ylabel('Package')
ax.set_title('优化结果')
ax.grid(True)
plt.show()
```