基于 0-1 规划的柔性印刷机优化方案设计

摘要

在柔性印刷中，频繁的切换墨盒会大大增加印刷的时间，降低印刷效率。因此本文针对包装顺序是否固定、墨盒卡槽顺序是否固定、以及考虑墨盒切换时间的情况下，将其转换成数学语言的不同的约束，建立最小化模型。

针对问题一，在固定包装顺序、恒定墨盒切换时间及无墨盒放置顺序约束的情况下，考虑以最小化总切换次数为目标，引入 0-1 变量 c_{jk} 和 x_{ijk}，分别表示包装 k 需要颜色 j，以及槽 i 在印刷包装 k 时需要颜色 j。将目标函数**线性化**，建立了在印刷约束、卡槽约束、墨盒约束等约束下最小化总切换次数的 0-1 **规划模型**。模型求解方面，采用 Gurobi 求解器进行求解，最终得到五个实例的解分别为 5、8、48、58、130，其中实例 1 具体分配方案详见图 1。

针对问题二，在问题一的基础上，考虑到不同墨盒的切换时间，新引入 0-1 变量 y_{ijk}，表示第 k 轮工作前是否将墨盒 i 替换为墨盒 j。建立了在色仓容量约束、墨盒需求约束、换色操作等约束下最小化总切换时间的 0-1 **规划模型**，模型求解方面，仍采用 Gurobi 求解器进行求解，最终得到四个实例的解分别为 32、51、111、202、326，其中实例 1 的具体分配方案详见图 3。

针对问题三，在墨盒装载顺序的约束的情况下，考虑将问题理解为一个**网络流**，将解抽象为在网络 N 上从源点到目的地的 K 条路径（路径访问特定网络 N 的不相交节点）、将网络流的弧映射为不同墨盒的切换成本、将目标函数抽象为 K 条路径的下最小总成本。建立了**流量守恒约束**、路径唯一性约束、顺序约束等约束下的 0-1 **规划模型**。模型求解方面仍采用 Gurobi 求解器进行求解。特别的，团队在预处理阶段建立了墨盒切换时间的**三角不等式**，考虑如何通过合并作业来简化问题，进而减少换墨盒操作的数量。结果方面，四个实例的解分别为 56、178、242、258，其中实例 1 的具体分配方案详见图 6。

针对问题四，在无包装顺序约束的情况下，考虑引入变量 O_{tk}，表示是否在第 t 个时间戳完成第 k 次包装。在问题三的基础上上新引入作业一致性的约束。建立了在流量守恒、路径唯一性、作业一致性等约束下最小化总切换时间的**非线性** 0-1 **规划模型**。模型求解方面仍采用 Gurobi 求解器进行求解，最终得到五个实例的解分别为 20、37、65、250，其中实例 3 的工作包装顺序为：7 6 1 5 3 4 2

关键词：**最优化理论　0-1 规划　三角不等式　网络流有向图**

一、问题背景与重述

1.1 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。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。但是在柔性版印刷时由于插槽数量有限，因此无法一次性的将所有墨盒全部放置到插槽当中。我们需要对包装种类、插槽、墨盒进行一定的规划，减少总切换时间从而提升印刷效率

1.2 要解决的具体问题

问题一、二、三、四都给出了包装种类编号、所需要的墨盒编号、卡槽序号。

1. 问题一中要求我们在给定包装种类印刷顺序的情况下，不考虑其他约束条件的情况下，建立总切换次数最小的模型，根据数据求解具体方案。

2. 问题二中要求我们在给定包装印刷顺序的情况下，考虑不同颜色墨盒切换时间情况下，建立总切换时间最小的模型，根据数据求解具体方案。

3. 问题三中我们要在给定包装印刷顺序的情况下，考虑不同颜色墨盒切换时间情况，不同包装对应的墨盒顺序是不同且固定的，我们在卡槽放置墨盒时需要对应不同包装正确的墨盒顺序，建立总切换时间最小的模型，根据数据求解具体方案。

4. 问题四在印刷之前要先确定包装种类的印刷顺序，并且不同颜色墨盒切换时间情况不同，不同包装对应的墨盒顺序是不同且固定的，在卡槽放置墨盒时需要对应不同包装正确的墨盒顺序，建立总切换时间最小的模型，根据数据求解具体方案。

二、问题分析

2.1 整体问题分析

通过对于题目的背景及问题的分析我们可以概括性的得出，问题一、二、三、四是在包装顺序是否固定，是否考虑墨盒切换时间，墨盒卡槽的放置顺序是否有要求的条件下进行考虑，我们通过整数规划将其转换成数学语言的约束条件，建立起最小值目标函数，从而进行求解。

2.2 基于不同问题分析

2.2.1 问题一分析

问题一给定的包装种类的印刷顺序，不考虑墨盒的切换时间，也不考虑墨盒卡槽的放置顺序，要求我们求出最小的切换次数。分析问题可得知当卡槽被分配到需要的墨盒和包装时则可正确工作，包装需要被喷上正确的颜色。因此我们引入 0-1 整数变量，来表达卡槽、包装是否正确分配，将问题转换成一个序列优化问题。

2.2.2 问题二分析

问题二与问题一的差别仅仅在于问题二在问题一的基础之上加入了墨盒切换的时间，因此我们在问题一的模型基础之上加入时间系数，在每次切换时计算切换时间，将总切换次数模型转换成总切换时间模型。

2.2.3 问题三分析

问题三因为考虑到不同包装对应的墨盒顺序是不同且固定的，所以在卡槽当中放置墨盒时需要对应不同的正确顺序。将不同的包装对应的墨盒顺序抽象成一个节点，引入图论中网络流的概念，通过弧流公式建立约束，得到模型。

2.2.4 问题四分析

问题四因为还需要考虑不同作业之间的顺序，我们引入变量 o_{ij} 用来代表作业实际的次序，然后与第三问中设置的变量 $u_{j_1,k_1,j_2,k_2,i}$ 做乘法，就能得到在真实次序下的表示墨盒颜色切换的变量。然后延续第三问的思路，依旧考虑相同的约束条件，最后得到相应的模型。

三、模型假设

1. 每一个卡槽只能插一个墨盒。
2. 一个墨盒只对应一个颜色，模型当中没有重复的颜色。
3. 工作轮数 K 代表的便是印刷包装 k

四、符号说明

符号	说明
c_{jk}	包装 k 需要 j 色时为 1，否则为 0
x_{ijk}	槽 i 需要印刷包装 k 需要 j 色时为 1，否则为 0
$Tans_1$	问题一中的总切换次数
$Tans_2$	问题二中的总切换时间
$Tans_3$	问题三中的总切换时间
$Tans_4$	问题四中的总切换时间
K	卡槽的最大容量
t_{ij}	从颜色 i 切换到颜色 j 所需要的时间
x_{jk}	印刷包装 k 时，装的 j 色时 $=1$，否则 $=0$
y_{ijk}	在 k 轮时把 i 色换成 j 色
$u_{j_1,k_1,j_2,k_2,i}$	：表示在 i 槽时，将 k_1 轮的 j_1 色转换成 j_2 色
C_k	k 轮所需的颜色
t_{j_1,j_2}	从颜色 j_1 转换成颜色 j_2 所需要的时间
o_mn	表示时间 m 完成作业 n，若完成则为 1，否则为 0

五、模型建立与求解

5.1 问题一模型建立与求解

5.1.1 决策变量

我们利用 0-1 整数规划引入变量。我们设

c_{jk} : 包装k需要j色时为 1，否则为 0

x_{ijk} : 槽i印刷包装k需要j色时为 1，否则为 0

5.1.2 目标函数

题目中给出我们要在给定包装顺序、不考虑不同墨盒切换时间、墨盒顺序可以任意放置的情况下求出能够完成印刷并且总切换次数最小的数学模型。

我们设目标函数为:

$$Tans = \frac{1}{2} min \sum_{k>1} \sum_{i} \sum_{j} |x_{i,j,k} - x_{i,j,k-1}| \tag{1}$$

其中目标函数表示的是如果对应卡槽上次颜色和这一次颜色不同，由于变量的 0-1 限制上一次的 $x_{i,j,k-1}$ 将会变成 0，这一次对应的 $x_{i,j,k}$ 则会变成 1，从而我们则可以将次数加 1，通过求和算出最小的目标函数。

由于目标函数中含有绝对值,我们使用线性化手段去除绝对值。我们设 $s_{i,j,k} = |x_{i,j,k} - x_{i,j,k-1}|$ 于是可以得到如下约束:

$$s_{i,j,k} \geq x_{i,j,k} - x_{i,j,k-1} \tag{2}$$

$$s_{i,j,k} \geq x_{i,j,k-1} - x_{i,j,k} \tag{3}$$

5.1.3 约束条件

基于题目中给出的要求的整理，我们问题一建立如下约束:

1. 印刷约束

我们要保证每一次印刷是每一个包装印刷的颜色都是齐全的，即:

$$\sum_{i} x_{ijk} \geq c_{jk}, \forall k, j \tag{4}$$

2. 卡槽约束

保证每次每个卡槽中只有一个颜色，即只有一个墨盒:

$$\sum_{j} x_{ijk} = 1, \forall k, i \tag{5}$$

3. 墨盒约束

在任意时刻时，每个颜色至多只能在一个卡槽出现:

$$\sum_{i} x_{ijk} \leq 1, \forall k, j \tag{6}$$

5.1.4 模型建立

综上决策变量、目标函数、约束条件的叙述，我们可以得到最小切换次数模型为

$$\min Tans_1 = \frac{1}{2}\sum_{k>1}\sum_{i}\sum_{j} s_{ijk}$$

$$\text{s.t.} \begin{cases} s_{i,j,k} \geq x_{i,j,k} - x_{i,j,k-1} \\ s_{i,j,k} \geq x_{i,j,k-1} - x_{i,j,k} \\ \sum_i x_{ijk} \geq c_{jk}, \forall k, j \\ \sum_j x_{ijk} = 1, \forall k, i \\ \sum_k x_{ijk} \leq K, \forall i, j \end{cases} \tag{7}$$

5.1.5 模型求解

具体公式已在附录 Python 代码中实现，使用 Gurobi 9.1.2 作为整数规划求解器。测试在 12 代 Intel(R) Core(TM) i7-12700H CPU 上执行，该 CPU 具有 14 个物理核心和 20 个逻辑处理器，使用了最多 20 个线程。该 CPU 支持 SSE2、AVX 和 AVX2 指令集。系统配备了 16 GB 内存。我们允许 Gurobi 使用所有可用线程。CPU 时间限制设置为 900 秒。下面是我们的求解流程伪代码

Algorithm 1 Branch and Bound for Solving Model 1

```
Input: Number of packages N, Number of colors M, Number of slots K, Package-color requirement matrix C
Output: Optimal color assignments and minimum switch count
Root Node:
root ← (level = 0, color_assignment = [], switches = 0)
Enqueue root into queue
Branch and Bound:
while queue ≠ ∅ do
    current_node ← dequeue(queue)
    if current_node.level = N then
        if current_node.switches < min_switches then
            Update best_solution
            min_switches ← current_node.switches
        end if
    else
        next_level ← current_node.level + 1
        for each color_combination in generate_color_combinations(K, M) do
            if is_valid_combination(color_combination, C[next_level]) then
                new_assignment ← current_node.color_assignment + color_combination
                new_switches ← calculate_switches(new_assignment, next_level)
                if new_switches < min_switches then
                    new_node ← (next_level, new_assignment, new_switches)
                    Enqueue new_node into queue
                end if
            end if
        end for
    end if
end while
=0
```

求解结果：

1. 得出其中总切换次数最小分别为 10、16、96、116、260。

2. 部分具体的卡槽分配方式如下图如所示，其他求解方案在附件中给出

附件一中 1_5_10_2 的具体方案:

Slot	1	2	1	2	1	2	1	2	1	2
Color	7	5	2	5	2	4	2	8	3	7
Packaging	1	1	2	2	3	3	4	4	5	5

图 1 问题一中附件 1_5_10_2 的具体方案

附件一中 2_7_10_3 的具体方案:

Slot	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
Color	2	7	3	8	6	3	9	6	7	9	6	7	4	6	7	2	6	7	3	1	7
Packaging	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7

图 2 问题一中附件 2_7_10_3 的具体方案

5.2 问题二模型建立与求解

5.2.1 决策变量

我们首先沿用问题一给出的 c_{jk} 变量，因为问题二在问题一的基础之上要求我们考虑不同墨盒切换时间，所以我们引入时间变量。同时我们对于问题一给出的 x_{ijk} 变量进行一定的改变，转换成 x_{jk}。

$$x_{jk}：印刷包装k时，装的j色时 = 1，否则 = 0$$

$$t_{ij}：从颜色i切换到颜色j所需要的时间$$

$$y_{ijk}：在k轮时把i色换成j色$$

因此我们可以知道问题二中的决策变量集为：

$$\{x_{jk}, y_{ijk}\}$$

5.2.2 目标函数

题目要求我们在给定包装顺序，考虑不同颜色墨盒切换时间情况下，即添加时间变量的情况下，建立数学模型，求解最小时间。

我们设目标函数为：

$$\min Tans_2 = \sum_{k>1} \sum_{j_2,j_1} t_{j_1,j_2} y_{j_1,j_2,k} \tag{8}$$

我们的目标函数对所有可能的 j_1 到 j_2 色转换乘以时间求和，于是能够得到切换颜色的时间总和。

5.2.3 约束条件

根据题目我们可以得到问题二的约束条件为

1. 保证每次印刷的颜色齐全

$$x_{jk} \geq c_{jk}, \forall k, j \tag{9}$$

2. 保证印刷时颜色总数小于总卡槽数

$$\sum_{j} x_{jk} \leq K, \forall k, j \tag{10}$$

3. 由于颜色切换存在三角不等式关系，即 i 色切换到 j 色所需时间，一定小于 i 色切换到 k 色再切换到 j 色的时间之和，因此我们可以知道，在 K 轮作业时我们要保证 i 色换成 j 色时，可得到

$$x_{j,k-1} = 1\text{或}x_{i,k-1} = 0\text{时，不能替换，}y_{ijk} = 0$$

$$x_{j,k-1} = 0\text{且}x_{i,k-1} = 1\text{时，可以替换}$$

基于此我们可以给出如下约束条件

$$y_{ijk} \leq 1 - x_{j,k-1} \tag{11}$$

$$y_{ijk} \leq x_{i,k-1} \tag{12}$$

$$\sum_{i} y_{ijk} \leq 1 \tag{13}$$

$$\sum_{j} y_{ijk} \leq 1 \tag{14}$$

$$x_{j,k} = x_{j,k-1} + \sum_{i} y_{ijk} - \sum_{i} y_{jik} \tag{15}$$

其中，第 1，2 条约束表示在 k-1 轮存在 j 色或者 k-1 轮不存在 i 色时，不能将 i 色转换为 j 色。第 3，4 条约束表明对于每一种颜色，至多只能由一种颜色转换而来或至多转换成另一种颜色。第 5 条约束表明 x_{jk} 的值取决于 $x_{j,k-1}$ 和在第 k 轮是否将其转换以及是否将其他颜色转换成它决定。

5.2.4 模型建立

综上决策变量、目标函数、约束条件的叙述，我们可以得到最小切换时间模型为

$$\min Tans_2 = \sum_{k>1}\sum_{j_2,j_1} t_{j_1,j_2} y_{j_1,j_2,k}$$

$$\text{s.t.} \begin{cases} x_{jk} \geq c_{jk}, \forall k, j \\ \sum_j x_{jk} \leq K, \forall k, j \\ y_{ijk} \leq 1 - x_{j,k-1} \\ y_{ijk} \leq x_{i,k-1} \\ \sum_i y_{ijk} \leq 1 \\ \sum_j y_{ijk} \leq 1 \\ x_{j,k} = x_{j,k-1} + \sum_i y_{ijk} - \sum_i y_{jik} \end{cases} \tag{16}$$

5.2.5 模型求解

利用 Gurobi 求解器进行求解，具体参数与问题一类似，下面给出我们的求解流程伪代码

Algorithm 2 Branch and Bound for Minimizing Total Switch Time

Require: Number of packages N, Number of colors M, Number of slots K, Package-color requirement matrix C, Color switching time matrix T
Ensure: Optimal color assignments and minimum switch time

```
best_solution ← ∅
min_time ← ∞
queue ← ∅
root ← (level = 0, color_assignment = ∅, time = 0)
Enqueue root into queue
while queue ≠ ∅ do
    current_node ← dequeue(queue)
    if current_node.level = N then
        if current_node.time < min_time then
            min_time ← current_node.time
            best_solution ← current_node.color_assignment
        end if
    else
        next_level ← current_node.level + 1
        for all color_combination ∈ generate_color_combinations(K, M) do
            if is_valid_combination(color_combination, C[next_level]) then
                new_assignment ← current_node.color_assignment ∪ color_combination
                new_time ← calculate_switch_time(new_assignment, next_level, T)
                if new_time < min_time then
                    new_node ← (next_level, new_assignment, new_time)
                    Enqueue new_node into queue
                end if
            end if
        end for
    end if
end while
return  best_solution, min_time =0
```

求解结果

1. 得出其中总切换时长最小分别为 32、51、111、202、326。
2. 我们在这里同问题一模型求解给出附件二实例一、二的具体卡槽分配方案。

附件二中实例一的具体方案:

Slot	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
Color	3	4	6	3	5	6	3	5	1	9	5	1	9	2	1
Packaging	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5

图 3 问题二中实例一的具体方案

附件二中实例二的具体方案:

Slot	1	2	1	2	1	2	1	2	1	2	1	2	1	2
Color	2	3	8	3	5	3	4	3	5	6	3	1	6	1
Packaging	1	1	2	2	3	3	4	4	5	5	6	6	7	7

图 4 问题二中实例二的具体方案

5.3 问题三模型建立与求解

5.3.1 有向图模型建立

通过对于问题三的分析，由于题目要求我们在进行打印时，卡槽中的墨盒顺序是不同且固定的，要去搜索在最小时间、固定墨盒顺序限制下的最优解，所以我们将问题抽象成一个图论当中的有向图搜索问题。

建立有向图 $G=(U,E)$, 其中 U 表示的是有向图中的节点，U 节点是一个二元变量为 (C_K,K), 其中 C_K 为 K 轮作业时墨盒在卡槽中的顺序，我们将 C_K 这个一元多维变量展开，将节点的网络流问题化成在给定的包装顺序下化成一个不同卡槽的网络流有向图问题。

下面我们基于附件数据格式给出的有向图帮助理解

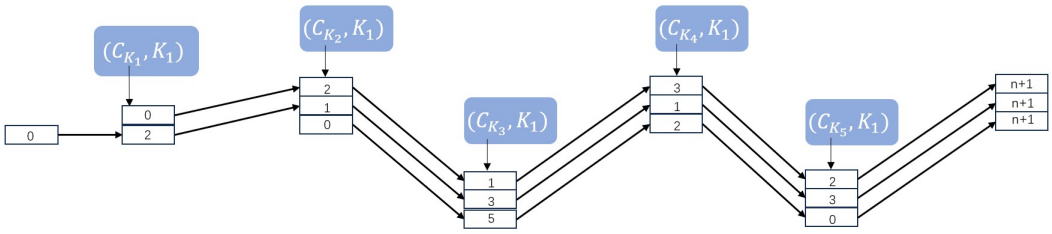

图 5 有向图呈现

在图中所表达的 0 节点是保证当前每一个墨盒卡槽都有前一个墨盒卡槽转入，即我们每一个卡槽都增添一个虚拟颜色 0。$n+1$ 节点则是保证当前每一个节点都可以转入下一个节点，其颜色也是虚拟颜色 0。

同时虽然我们只给出了卡槽按照包装顺序的通路，但是假设节点 m 与节点 n 之间有一个节点 k，在节点 m 变换到节点 k 后不需要再次变换墨盒顺序就可以到达节点 n 的话，其节点 m、n 之间应该也有一条通路

5.3.2 决策变量

我们给予每一条边一个边权，用来表达边的两端是否是联通的，即在槽 i 中是否能够从当前颜色转换成下一个颜色

$u_{j_1,k_1,j_2,k_2,i}$：表示在i槽时，将k_1轮的j_1色转换成j_2色

K：总轮数

C_k：k 轮所需要的颜色

我们沿用问题二中墨盒切换的时间变量

t_{j_1,j_2}：从颜色j_1切换到颜色j_2所需要的时间

5.3.3 目标函数

设目标函数为

$$\min\ Trans_3 = \sum_{i}\sum_{k_2,k_1}\sum_{j_2,j_1} t_{j_2,j_1} u_{j_1,k_1,j_2,k_2,i} \tag{17}$$

5.3.4 约束条件

1. 我们要保证每一次进行包装印刷时，卡槽中能够有对应的墨盒顺序，即我们要保证在每一轮包装印刷前卡槽中能够有该种颜色（允许从 0 色转化）

$$\sum_{j_1\in C_{k_1},k_1<k_2}\ \sum_{i} u_{j_1,k_1,j_2,k_2,i} = 1, \forall k_2, j_2 \in C_{K_2} \tag{18}$$

2. 我们需要保证所有颜色最终都成功转化成了别的颜色（允许转化成 0 色）

$$\sum_{j_1\in C_{k_1},k_1>k_2}\ \sum_{i} u_{j_1,k_1,j_2,k_2,i} = 1, \forall k_2, j_2 \in C_{K_2} \tag{19}$$

3. 我们要保证每一个颜色，其来源（转化来时的颜色）以及去向（转化成的其他颜色）只在同一个槽上进行，即在有向图中每一个点不能由两条不同的槽对应的通路通过，也就是不能出现一个节点上面有两条同方向的通路

$$\sum_{j_1,k_1<k_2} u_{j_1,k_1,j_2,k_2,i} = \sum_{j_1,k_1>k_2} u_{j_1,k_1,j_2,k_2,i}\ \forall i, k_2, j_2 \in C_{k_2} \tag{20}$$

4. 我们要保证每一个卡槽的墨盒颜色都是从我们所设的虚拟颜色 0 开始更新

$$\sum_{j_1\in C_{k_1},k_1} u_{0,0,j,k,i} = 1, \forall i \tag{21}$$

5. 保证我们每一个卡槽的颜色都能够转入最终转入虚拟节点

$$\sum_{j_1 \in C_{k_1}, k_1} u_{j,k,0,K+1,i} = 1, \forall i \tag{22}$$

6. 保证顺序

$$\sum_{j_1 \in C_{k_1}, k_1 < k_2} \sum_{i < i_1} u_{j_1,k_1,j_{21},k_2,i} \geq \sum_{j_1 \in C_{k_1}, k_1 < k_2} \sum_{i < i_1} u_{j_1,k_1,j_{22},k,i} \forall i_1, k_2, j_{21}, j_{22} \in C_{k_2} \tag{23}$$

其中 j_{21}, j_{22} 是 C_{k_2} 所有两两相邻的颜色。为了保证其在卡槽中的顺序，我们只需要保证对于前 i_1 个卡槽，j_{22} 出现时,j_{21} 一定出现过即可。

结合目标函数和约束条件我们可以得出整个最小切换时间模型为

$$\begin{aligned}
&\min Trans_3 = \sum_{k_2,k_1} \sum_{j_2,j_1} \sum_{i} t_{j_2,j_1} u_{j_1,k_1,j_2,k_2,i} \\
&\text{s.t.} \begin{cases}
\sum_{j_1 \in C_{k_1}, k_1 < k_2} \sum_i u_{j_1,k_1,j_2,k_2,i} = 1, \forall k_2, j_2 \in C_{K_2} \\
\sum_{j_1 \in C_{k_1}, k_1 > k_2} \sum_i u_{j_1,k_1,j_2,k_2,i} = 1, \forall k_2, j_2 \in C_{K_2} \\
\\
\sum_{j_1, k_1 < k_2} u_{j_1,k_1,j_2,k_2,i} \\
= \sum_{j_1, k_1 > k_2} u_{j_1,k_1,j_2,k_2,i} \forall i, k_2, j_2 \in C_{k_2} \\
\\
\sum_{j_1 \in C_{k_1}, k_1} u_{0,0,j,k,i} = 1, \forall i \\
\sum_{j_1 \in C_{k_1}, k_1} u_{j,k,0,K+1,i} = 1, \forall i \\
\\
\sum_{j_1 \in C_{k_1}, k_1 < k_2} \sum_{i < i_1} u_{j_1,k_1,j_{21},k_2,i} \\
\geq \sum_{j_1 \in C_{k_1}, k_1 < k_2} \sum_{i < i_1} u_{j_1,k_1,j_{22},k,i} \forall i_1, k_2, j_{21}, j_{22} \in C_{k_2}
\end{cases}
\end{aligned} \tag{24}$$

5.3.5 模型求解

利用 Gurobi 求解器进行求解，具体参数与问题一类似，下面给出我们的求解流程伪代码

Algorithm 3 Branch and Bound for Minimizing Total Switch Time

Require: Number of packages N, Number of colors M, Number of slots K, Package-color requirement matrix C, Color switching time matrix T
Ensure: Optimal color assignments and minimum switch time

best_solution $\leftarrow \emptyset$
min_time $\leftarrow \infty$
queue $\leftarrow \emptyset$
root $\leftarrow$ (level $= 0$, color_assignment $= \emptyset$, time $= 0$)
Enqueue root into queue
while queue $\neq \emptyset$ do
  current_node $\leftarrow$ dequeue(queue)
  if current_node.level $= N$ then
    if current_node.time $<$ min_time then
      min_time $\leftarrow$ current_node.time
      best_solution $\leftarrow$ current_node.color_assignment
    end if
  else
    next_level $\leftarrow$ current_node.level $+ 1$
    for all color_combination $\in$ generate_color_combinations(K, M) do
      if is_valid_combination(color_combination, C[next_level]) then
        new_assignment $\leftarrow$ current_node.color_assignment $\cup$ color_combination
        new_time $\leftarrow$ calculate_switch_time(new_assignment, next_level, T)
        if new_time $<$ min_time then
          new_node $\leftarrow$ (next_level, new_assignment, new_time)
          Enqueue new_node into queue
        end if
      end if
    end for
  end if
end while
return best_solution, min_time
return true =0

求解结果：

1. 得出其中总切换时长最小分别为 56、178、242、258
2. 下面我们给出在问题三模型求解下的部分具体分配方案

Slot	1	2	1	2	1	2	1	2	1	2
Color	3	2	2	1	1	3	3	1	2	3
Packagin	1	1	2	2	3	3	4	4	5	5

图 6 问题三中实例一的具体方案

Slot	3	5	6	7	8	9	10	4	5	6	7	8	9	2
Color	29	23	5	25	8	28	22	6	26	15	9	12	23	22
Packagin	1	1	1	1	1	1	1	2	2	2	2	2	2	3

图 7 问题三中实例二的具体方案前半部分

5.4 问题四模型建立与求解

问题四要求我们在问题三的基础之上考虑包装的顺序约束，建立最小时间值模型，因此我们在问题三的基础有向图不变的情况下进行一定的变化。有向图每一个节点代表一个包装，问题四要求包装顺序随机，因此我们将节点变成一个三元变量 $((C_K, K, t))$，

其中 t 为我们加入的新变量，用一个真实的递增的时间戳表示工作顺序。因此在我们展开的有向图中我们便可以在每一个卡槽节点加上时间约束，在时间约束的基础之上求最小值函数。

由于问题四中包装的顺序并不是固定的，所以有向图并不是像问题三中的一条一定从包装 1 开始的链，而是可能从别的节点开始搜索，下面我们给出问题四其中一种可能的有向图情况帮助我们进行理解。

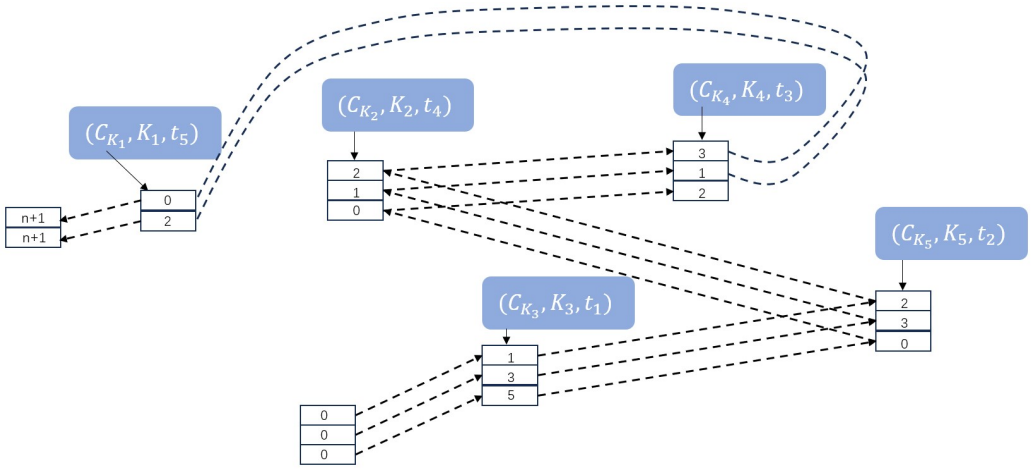

图 8 有向图呈现

在该图中我们将节点 $((C_{K_3}, K_3, t_1))$ 作为开始的节点，并将三元变量中的 t 元素下标赋值为 1，表示这个是我们所搜索或者遍历的第一个节点。其中有向图的通路表示搜索的流程，在求解过程中不同的实例其最小化时间的通路是不同的。

同样的在图中所表达的 0 节点是保证当前每一个墨盒卡槽都有前一个墨盒卡槽转入，即我们每一个卡槽都增添一个虚拟颜色 0。$n+1$ 节点则是保证当前每一个节点都可以转入下一个节点，其颜色也是虚拟颜色 0。

同时虽然我们只给出了卡槽按照包装顺序的通路，但是假设节点 m 与节点 n 之间有一个节点 k，在节点 m 变换到节点 k 后不需要再次变换墨盒顺序就可以到达节点 n 的话，其节点 m、n 之间应该也有一条通路

5.4.1 决策变量

我们首先可以沿用问题三给出的决策变量 $\{u_{j_1,k_1,j_2,k_2,i}, K\}$，然后我们设置一个新的决策变量

o_{mn}：表示时间m完成作业n，若完成则为 1，否则为 0

综上我们可以得到问题四的决策变量集为

$$\{u_{j_1,k_1,j_2,k_2,i}, o_{mn}\}$$

5.4.2 目标函数

将目标函数设为

$$\min Trans_4 = \sum_{k_2,j_2 \in C_{k_2}} \sum_{k_1,j_1 \in C_{k_1}} \sum_i t_{j_2,j_1} o_{t_1 k_1} o_{t_2 k_2} u_{j_1,k_1,j_2,k_2,i} \tag{25}$$

5.4.3 约束条件

1. 我们在问题四中同样要保证每一次进行包装印刷时，卡槽中能够有对应的墨盒顺序，即我们要保证在每一轮包装印刷前卡槽中能够有该种颜色，但是由于我们引入了递增的真实的时间戳 t, 所以我们要对于该约束进行改变

$$\sum_{t_2} \sum_{t_1<t_2} \sum_{j_1,k_1} \sum_i o_{t_1,k_1} o_{t_2,k_2} u_{j_1,k_1,j_2,k_2,i} = 1, \forall k_2, j_2 \in C_{K_2} \tag{26}$$

我们注意到内含 0-1 变量的乘法运算，因此我们采取线性化手段 (以下出现乘法的地方我们做相同处理): 我们令 $ou_{t_1,t_2,k_1,k_2,j_1,j_2,i} = o_{t_1,k_1} o_{t_2,k_2} u_{j_1,k_1,j_2,k_2,i}$, 于是可以得到如下约束:

$$ou_{t_1,t_2,k_1,k_2,j_1,j_2,i} \leq o_{t_1,k_1} \tag{27}$$

$$ou_{t_1,t_2,k_1,k_2,j_1,j_2,i} \leq o_{t_2,k_2} \tag{28}$$

$$ou_{t_1,t_2,k_1,k_2,j_1,j_2,i} \leq u_{j_1,k_1,j_2,k_2,i} \tag{29}$$

$$ou_{t_1,t_2,k_1,k_2,j_1,j_2,i} \geq o_{t_1,k_1} + o_{t_2,k_2} + u_{j_1,k_1,j_2,k_2,i} - 2 \tag{30}$$

2. 我们给定的约束 1 保证了在印刷前肯定有墨盒对应下一轮需要的卡槽，因此我们要保证在印刷时每一次颜色都能够转化成别的别的轮次的颜色

$$\sum_{t_2} \sum_{t_1>t_2} \sum_{j_1,k_1} \sum_i o_{t_1,k_1} o_{t_2,k_2} u_{j_2,k_2,j_1,k_1,i} = 1, \forall k_2, j_2 \in C_{K_2} \tag{31}$$

3. 总是我们的包装顺序是不定的，但是我们也要保证每一个槽在变换时只在当前槽上进行变换，即在有向图中边不能够有交点，同时也不能出现一个节点上面有两条同方向的通路

$$\sum_{t_2} \sum_{t_1<t_2} \sum_{j_1,k_1} o_{t_1,k_1} o_{t_2,k_2} u_{j_1,k_1,j_2,k_2,i}$$

$$= \sum_{t_2} \sum_{t_1>t_2} \sum_{j_1,k_1} o_{t_1,k_1} o_{t_2,k_2} u_{j_2,k_2,j_1,k_1,i} = 1, \forall k_2, j_2 \in C_{K_2} \tag{32}$$

4. 我们要保证每一个卡槽的墨盒颜色都是从我们所设的虚拟颜色 0 开始更新

$$\sum_{j_1 \in C_{k_1},k_1} u_{0,0,j,k,i} = 1, \forall i \tag{33}$$

5. 保证我们每一个节点都能够转入下一个节点，即能从当前的颜色转换成虚拟颜

色 0

$$\sum_{j_1 \in C_{k_1}, k_1} u_{j,k,0,K+1,i} = 1, \forall i \tag{34}$$

6. 保证顺序约束, 这里变量的含义与第三问中顺序约束的变量含义完全相同。

$$\sum_{t_2} \sum_{t_1<t_2} \sum_{j_1,k_1} \sum_{i<i_1} o_{j_1,k_1} o_{j_2,k_2} u_{j_1,k_1,j_{21},k_2,i}$$

$$\geq \sum_{t_2} \sum_{t_1<t_2} \sum_{j_1,k_1} \sum_{i<i_1} o_{j_1,k_1} o_{j_2,k_2} u_{j_1,k_1,j_{22},k_2,i} = 1, \forall i \tag{35}$$

7. 我们设置了 o_{mn} 决策变量来限制作业的顺序，所以我们要将该决策变量约束从而保证作业顺序为正确的。因为 o_{mn} 决策变量为 0-1 变量所以我们可以建立以下约束

$\sum_m o_{mn} = 1$: 对于任意时间来说，对应的包装，即作业轮次只能有 1 个

$\sum_n o_{mn} = 1$: 对于任意包装，即轮次来说，对应时间只能有 1 个

5.4.4 模型建立

结合目标函数和约束条件我们可以得出整个最小切换时间模型为

$$\min Trans_4 = \sum_{k_2, j_2 \in C_{k_2}} \sum_{k_1, j_1 \in C_{k_1}} \sum_i t_{j_2,j_1} o_{t_1 k_1} o_{t_2 k_2} u_{j_1,k_1,j_2,k_2,i}$$

$$\text{s.t.} \begin{cases} \sum_{t_2} \sum_{t_1<t_2} \sum_{j_1,k_1} \sum_i o_{t_1,k_1} o_{t_2,k_2} u_{j_1,k_1,j_2,k_2,i} = 1, \forall k_2, j_2 \in C_{K_2} \\ \sum_{t_2} \sum_{t_1>t_2} \sum_{j_1,k_1} \sum_i o_{t_1,k_1} o_{t_2,k_2} u_{j_2,k_2,j_1,k_1,i} = 1, \forall k_2, j_2 \in C_{K_2} \\ \sum_{t_2} \sum_{t_1<t_2} \sum_{j_1,k_1} o_{t_1,k_1} o_{t_2,k_2} u_{j_1,k_1,j_2,k_2,i} \\ = \sum_{t_2} \sum_{t_1>t_2} \sum_{j_1,k_1} o_{t_1,k_1} o_{t_2,k_2} u_{j_2,k_2,j_1,k_1,i} = 1, \forall k_2, j_2 \in C_{K_2} \\ \sum_{j_1 \in C_{k_1}, k_1} u_{0,0,j,k,i} = 1, \forall i \\ \sum_{j_1 \in C_{k_1}, k_1} u_{j,k,0,K+1,i} = 1, \forall i \\ \sum_{t_2} \sum_{t_1<t_2} \sum_{j_1,k_1} \sum_{i<i_1} o_{j_1,k_1} o_{j_2,k_2} u_{j_1,k_1,j_{21},k_2,i} \\ \geq \sum_{t_2} \sum_{t_1<t_2} \sum_{j_1,k_1} \sum_{i<i_1} o_{j_1,k_1} o_{j_2,k_2} u_{j_1,k_1,j_{22},k_2,i} = 1, \forall i \\ \sum_m o_{mn} = 1 \\ \sum_n o_{mn} = 1 \end{cases}$$

5.4.5 模型求解

利用 Gurobi 求解器求解，参数与问题一类似，下面给出我们的求解流程伪代码

Algorithm 4 Branch and Bound for Minimizing Total Switch Time with Order Constraints

Require: Number of packages N, Number of colors M, Number of slots K, Package-color requirement matrix C, Color switching time matrix T
Ensure: Optimal color assignments and minimum switch time

root $\leftarrow$ (level $= 0$, color_assignment $= \emptyset$, time $= 0$, timestamp $= 0$)
while queue $\neq \emptyset$ do
  current_node $\leftarrow$ dequeue(queue)
  if current_node.level $= N$ then
    if current_node.time $<$ min_time then
      min_time $\leftarrow$ current_node.time
      best_solution $\leftarrow$ current_node.color_assignment
    end if
  else
    next_level $\leftarrow$ current_node.level $+ 1$
    next_timestamp $\leftarrow$ current_node.timestamp $+ 1$
    for all color_combination $\in$ generate_color_combinations(K, M) do
      if is_valid_combination(color_combination, C[next_level], next_timestamp) then
        new_assignment $\leftarrow$ current_node.color_assignment $\cup$ color_combination
        new_time $\leftarrow$ calculate_switch_time(new_assignment, next_level, T)
        if new_time $<$ min_time then
          new_node $\leftarrow$ (next_level, new_assignment, new_time, next_timestamp)
          Enqueue new_node into queue
        end if
      end if
    end for
  end if
end while=0

求解结果：

1. 得出其中总切换时长最小分别为 20、37、65、250

2. 下面是问题四示例一在我们所建立的模型上面得到的具体分配方案

Slot	1	2	1	2	1	2	1	2	1	2
Color	4	2	7	3	7	8	7	5	2	5
Packaging	3	3	5	5	4	4	1	1	2	2

图 9 问题四中实例一的具体方案

六、模型的评价、改进和推进

6.1 模型评价

优点:

1. 引入 0-1 变量，使用线性化手段，将问题简化为线性规划模型，能够求得最优解

2. 参考了网络流模型，将各个作业所需墨盒抽象为节点，能够使用网络流算法求解问题

3. 模型不仅考虑墨盒切换时间，还考虑了卡槽物理限制、墨盒安装顺序等实际问题

缺点：

1. 只考虑了一台柔性印刷机的情形，在实际生产当中可能会有更多的柔性印刷机

2. 简单的用一个时间矩阵替代各个墨盒之间切换的代价，应当引入材料损失等更多代价

6.2 模型推进

1. 考虑多台柔性印刷机的情况

在实际生产中，通常会有多台柔性印刷机同时工作。因此，可以扩展模型以考虑多台印刷机的情况。具体实现可以通过引入更多的变量和约束来代表每台印刷机的状态和作业分配。我们可以通过引入更多变量，和建立更多约束来解决。

2. 考虑更多的切换成本

不仅考虑时间矩阵，还应引入材料损失、能耗等多种切换代价，以更加真实地模拟实际生产中的切换成本，引入材料损失和引入能耗成本。

3. 优化调度算法

可以考虑采用更高效的启发式算法，如遗传算法、粒子群算法等，结合线性规划模型进行求解，以提高求解速度和准确性。

七、参考文献

[1] 肖晓伟, 肖迪, 林锦国, 等. 多目标优化问题的研究概述 [J]. 计算机应用研究,2011,28(03):805-808+827.

[2] 苏永涛, 仉俊峰. 基于图论方法的路径规划应用 [J]. 电测与仪表,2012,49(01):94-96.

附录

附录 1：支撑材料的文件列表

附录 2：初始化代码和数据处理代码

```
%模型一的代码
import pulp as pl
import pandas as pd
file_name = 'Ins5_30_60_10.xlsx'
# 加载数据
packages_data = pd.read_excel(file_name, sheet_name='包装-墨盒-插槽')
needs_data = pd.read_excel(file_name, sheet_name='包装种类及其所需墨盒')

# 墨盒需求字典，每种包装需要哪些墨盒
needs_dict = {row['包装种类编号']: eval(row['所需墨盒编号']) for index, row in
    needs_data.iterrows()}# 创建一个最小化问题
model = pl.LpProblem("墨盒切换优化", pl.LpMinimize)

# 获取总的包装数、墨盒数和插槽数
num_packages = int(needs_data['包装种类编号'].max())
num_colors = int(max(max(needs) for needs in needs_dict.values()))
num_slots = int(packages_data['插槽序号'].max())

# 创建变量
x = pl.LpVariable.dicts("x",
                        ((slot, color, package) for slot in range(1, num_slots + 1)
                                                for color in range(1,
                                                    num_colors + 1)
                                                for package in range(1,
                                                    num_packages + 1)),
                        cat='Binary')

y = pl.LpVariable.dicts("y",
                        ((slot, color, package) for slot in range(1, num_slots + 1)
                                                for color in range(1,
                                                    num_colors + 1)
                                                for package in range(2,
                                                    num_packages + 1)),
```

```
                                cat='Binary')

# 确保每个包装中所需的每种墨盒都至少在一个插槽中
for package, colors in needs_dict.items():
    for color in colors:
        model += pl.lpSum(x[(slot, color, package)] for slot in range(1, num_slots
            + 1)) >= 1, f"ColorFulfillment_P{package}_C{color}"

# 每个插槽在每次包装时只能有一个颜色
for package in range(1, num_packages + 1):
    for slot in range(1, num_slots + 1):
        model += pl.lpSum(x[(slot, color, package)] for color in range(1,
            num_colors + 1)) == 1, f"OneColorPerSlot_S{slot}_P{package}"

# 添加约束以确保 y 为 x 的乘积
for slot in range(1, num_slots + 1):
    for color in range(1, num_colors + 1):
        for package in range(2, num_packages + 1):
            model += y[(slot, color, package)] >= x[(slot, color, package)] -
                x[(slot, color, package - 1)]
            model += y[(slot, color, package)] >= x[(slot, color, package - 1)
                ] - x[(slot, color, package)]

# 设置目标函数为最小化 y 的和
model += pl.lpSum(y[(slot, color, package)]
                  for slot in range(1, num_slots + 1)
                  for color in range(1, num_colors + 1)
                  for package in range(2, num_packages + 1)), "
                      Minimize_Transitions"

# 求解问题
model.solve(pl.GUROBI_CMD())

# 输出结果
for variable in model.variables():
    if variable.varValue > 0.5:
        print(f"{variable.name} = {variable.varValue}")
```

```
# 检查问题状态
print("Status:", pl.LpStatus[model.status])

%模型二的代码
import gurobipy as gp
from gurobipy import GRB
import pandas as pd
import numpy as np

# 加载 Excel 文件
filename = 'Ins1_5_10_3.xlsx'

# 读取数据
sheet1 = pd.read_excel(filename, sheet_name=0)
sheet2 = pd.read_excel(filename, sheet_name=1)
sheet3 = pd.read_excel(filename, sheet_name=2) # 切换时间矩阵

# 获取数据的维度，并转换为整数
num_packaging_types = int(sheet1.iloc[:, 0].max()) # 包装种类数量
num_colors = int(sheet1.iloc[:, 1].max()) # 墨盒（颜色）数量
num_slots = int(sheet1.iloc[:, 2].max()) # 从 sheet1 的第三列获取插槽数量
print(f'Packaging Types: {num_packaging_types}, Colors: {num_colors}, Slots:
    {num_slots}')
input( 'Press Enter to continue...' )
# 初始化 c 矩阵
c = np.zeros((num_packaging_types, num_colors), dtype=int) # 初始化矩阵 cjk

# 填充 c 矩阵
for index, row in sheet2.iterrows():
    packaging_type = int(row[0]) # 获取包装种类编号
    colors_needed = eval(row[1]) # 获取所需的墨盒编号，转换为数组
    for j in colors_needed:
        c[packaging_type-1, j-1] = 1 # 对应位置赋值为1

# 读取切换时间矩阵并确保其为数值类型
switch_time = sheet3.values
```

```
# 创建模型
model = gp.Model("color_switching")

# 定义决策变量
x = model.addVars(num_colors, num_packaging_types, vtype=GRB.BINARY,
    name="x")
y = model.addVars(num_colors, num_colors, num_packaging_types, vtype=GRB
    .BINARY, name="y")

# 约束条件
# 1. 保证每次印刷颜色齐全
for k in range(num_packaging_types):
    for j in range(num_colors):
        model.addConstr(x[j, k] >= c[k, j], name=f"color_full_{k}_{j}")

# 2. 保证颜色总数小于卡槽总数
for k in range(num_packaging_types):
    model.addConstr(gp.quicksum(x[j, k] for j in range(num_colors)) <=
        num_slots, name=f"slots_limit_{k}")

# 3. 在轮转时，如果替换为 j，要保证两个条件：
for k in range(1, num_packaging_types):
    for i in range(num_colors):
        for j in range(num_colors):
            model.addConstr(y[i, j, k] <= 1 - x[j, k-1], name=f"
                switch_constraint_1_{i}_{j}_{k}")
            model.addConstr(y[i, j, k] <= x[i, k-1], name=f"
                switch_constraint_2_{i}_{j}_{k}")

# 新的约束条件
# 4. \sum_i y_{ijk} \leq 1
for k in range(num_packaging_types):
    for j in range(num_colors):
        model.addConstr(gp.quicksum(y[i, j, k] for i in range(num_colors)) <= 1,
            name=f"sum_y_i_{j}_{k}")
```

```
# 5. \sum_j y_{ijk} \leq 1
for k in range(num_packaging_types):
    for i in range(num_colors):
        model.addConstr(gp.quicksum(y[i, j, k] for j in range(num_colors)) <= 1,
            name=f"sum_y_j_{i}_{k}")

# 6. x_{jk} = x_{j,k-1} + \sum_i y_{ijk} - \sum_i y_{ijk}
for k in range(1, num_packaging_types):
    for j in range(num_colors):
        model.addConstr(x[j, k] == x[j, k-1] + gp.quicksum(y[i, j, k-1] for i in
            range(num_colors)) - gp.quicksum(y[j, i, k] for i in range(num_colors
            )), name=f"x_balance_{j}_{k}")

# 目标函数：最小化总的换色时间
objective = gp.quicksum(switch_time[i, j] * y[i, j, k-1] for k in range(1,
    num_packaging_types) for i in range(num_colors) for j in range(num_colors)
    if i < switch_time.shape[0] and j < switch_time.shape[1])
model.setObjective(objective, GRB.MINIMIZE)

# 求解模型
model.optimize()

# 检查求解状态
if model.status == GRB.OPTIMAL:
    print('Optimal solution found:')
    x_value = model.getAttr('x', x)
    y_value = model.getAttr('x', y)

    # 显示结果
    print('x_value:')
    for k in range(num_packaging_types):
        for j in range(num_colors):
            if x_value[j, k] > 0.5:
                print(f'PackagingType {k+1}, Color {j+1}')

    print('y_value:')
    for k in range(1, num_packaging_types):
```

```
            for i in range(num_colors):
                for j in range(num_colors):
                    if y_value[i, j, k] > 0.5:
                        print(f'Step {k}, ColorFrom {i+1}, ColorTo {j+1}')
else:
    print('Solver failed to find an optimal solution.')


%模型三的代码
import pulp as pl
import pandas as pd
import numpy as np
import itertools
from itertools import pairwise
file_name = 'Ins1_5_5_2.xlsx'
# 加载数据
packages_data = pd.read_excel(file_name, sheet_name='包装-墨盒-插槽')
needs_data = pd.read_excel(file_name, sheet_name='包装种类及其所需墨盒')
time_table = pd.read_excel(file_name, sheet_name='墨盒切换时间', index_col
    =0).to_numpy()
# 墨盒需求字典，每种包装需要哪些墨盒
needs_dict = {row['包装种类编号']: eval(row['所需墨盒编号']) for index, row in
    needs_data.iterrows()}
M = 50
# 创建一个最小化问题
model = pl.LpProblem("墨盒切换优化", pl.LpMinimize)

# 获取总的包装数、墨盒数和插槽数
num_packages = int(needs_data['包装种类编号'].max())
num_colors = int(max(max(needs) for needs in needs_dict.values()))
num_slots = int(packages_data['插槽序号'].max())

needs_dict[0] = [0]
needs_dict[num_packages + 1] = [0]

# 创建变量
u = pl.LpVariable.dicts("u",((color1, s, color2, t, slot) for slot in range(1,
```

```
    num_slots + 1)
                                                    for s in range(num_packages +
                                                        1)
                                                    for t in range(1, num_packages
                                                        + 2)
                                                    for color1 in needs_dict[s]
                                                    for color2 in needs_dict[t]),
                            cat='Binary')

# 保证每个颜色被转化进来
for t in range(1, num_packages + 1):
    for color2 in needs_dict[t]:
        model += pl.lpSum(u[(color1, s, color2, t, i)] for s in range(t) for color1 in
            needs_dict[s] for i in range(1, num_slots + 1)) == 1

# 保证每个颜色被转化出去
for s in range(1, num_packages + 1):
    for color1 in needs_dict[s]:
        model += pl.lpSum(u[(color1, s, color2, t, i)] for t in range(s+1,
            num_packages+2) for color2 in needs_dict[t] for i in range(1,
            num_slots + 1)) == 1

# 保证颜色转入转出是同一个槽
for slot in range(1, num_slots):
    for time in range(1, num_packages + 1):
        for color in needs_dict[time]:
            model += pl.lpSum(u[(color1, s, color, time, slot)] for s in range(time)
                for color1 in needs_dict[s]) \
            == pl.lpSum(u[(color, time, color2, t, slot)] for t in range(time + 1,
                num_packages + 2) for color2 in needs_dict[t])

# 保证顺序
for t in range(1, num_packages + 1):
    for color1, color2 in pairwise(needs_dict[t]):
        for max_slot in range(1, num_slots + 1):
            model += pl.lpSum(u[(color3, s, color1, t, i)] for s in range(t) for
                color3 in needs_dict[s] for i in range(1, max_slot + 1)) >= \
```

```
                    pl.lpSum(u[(color3, s, color2, t, i)] for s in range(t) for color3
                        in needs_dict[s] for i in range(1, max_slot + 1))
# 每条路从0转出1个
for slot in range(1, num_slots + 1):
    model += pl.lpSum(u[(0, 0, color2, t, slot)] for t in range(1, num_packages +
        2) for color2 in needs_dict[t]) == 1

# 每条路最后转化到1个
for slot in range(1, num_slots + 1):
    model += pl.lpSum(u[(color1, s, 0, num_packages + 1, slot)] for s in range(
        num_packages + 1) for color1 in needs_dict[s]) == 1

# 目标函数
model += pl.lpSum(time_table[color1 - 1][color2 - 1] * u[(color1, s, color2, t,
    slot)]
                  for slot in range(1, num_slots + 1)
                  for s in range(1, num_packages)
                  for t in range(s + 1, num_packages + 1)
                  for color1 in needs_dict[s]
                  for color2 in needs_dict[t])


# 求解问题
model.solve(pl.GUROBI_CMD())

# 输出结果
for variable in model.variables():
    if variable.varValue > 0:
        print(f"{variable.name} = {variable.varValue}")

# 检查问题状态
print("Status:", pl.LpStatus[model.status])


%模型四的代码
import pulp as pl
import pandas as pd
```

```
import numpy as np
file_name = 'Ins2_5_10_3.xlsx'
# 加载数据
packages_data = pd.read_excel(file_name, sheet_name='包装-墨盒-插槽')
needs_data = pd.read_excel(file_name, sheet_name='包装种类及其所需墨盒')
time_table = pd.read_excel(file_name, sheet_name='墨盒切换时间', index_col
    =0).to_numpy()
# 墨盒需求字典，每种包装需要哪些墨盒
needs_dict = {row['包装种类编号']: eval(row['所需墨盒编号']) for index, row in
    needs_data.iterrows()}
import pulp as pl
from itertools import pairwise
# 创建一个最小化问题
model = pl.LpProblem("墨盒切换优化", pl.LpMinimize)

# 获取总的包装数、墨盒数和插槽数
num_packages = int(needs_data['包装种类编号'].max())
num_colors = int(max(max(needs) for needs in needs_dict.values()))
num_slots = int(packages_data['插槽序号'].max())

needs_dict[0] = [0]
needs_dict[num_packages + 1] = [0]

# 创建变量
u = pl.LpVariable.dicts("u",((color1, s, color2, t, slot) for slot in range(1,
    num_slots + 1)
                                                    for s in range(num_packages +
                                                        2)
                                                    for t in range(num_packages +
                                                        2)
                                                    for color1 in needs_dict[s]
                                                    for color2 in needs_dict[t]),
                          cat='Binary')

o = pl.LpVariable.dicts("o",((s,t) for s in range(num_packages + 2)
                                   for t in range(num_packages + 2)),
                          cat='Binary')
```

```
ou = pl.LpVariable.dicts("ou",((t1,t2, color1, s, color2, t, slot)
                                for t1 in range(num_packages + 2)
                                for t2 in range(num_packages + 2)
                                for slot in range(1, num_slots + 1)
                                for s in range(num_packages + 2)
                                for t in range(num_packages + 2)
                                for color1 in needs_dict[s]
                                for color2 in needs_dict[t]),
                          cat='Binary')

uo = pl.LpVariable.dicts("uo",((t1,t2, color1, s, color2, t, slot)
                                for t1 in range(num_packages + 2)
                                for t2 in range(num_packages + 2)
                                for slot in range(1, num_slots + 1)
                                for s in range(num_packages + 2)
                                for t in range(num_packages + 2)
                                for color1 in needs_dict[s]
                                for color2 in needs_dict[t]),
                          cat='Binary')
# 定义乘积
for t1 in range(num_packages + 2):
    for t2 in range(num_packages + 2):
        for slot in range(1, num_slots + 1):
            for s in range(num_packages + 2):
                for t in range(num_packages + 2):
                    for color1 in needs_dict[s]:
                        for color2 in needs_dict[t]:
                            model += ou[(t1, t2, color1, s, color2, t, slot)] <= o[(
                                t1,t)]
                            model += ou[(t1, t2, color1, s, color2, t, slot)] <= o[(
                                t2,s)]
                            model += ou[(t1, t2, color1, s, color2, t, slot)] <= u
                                [(color1, s, color2, t, slot)]
                            model += ou[(t1, t2, color1, s, color2, t, slot)] >= o[(
                                t1,t)] + o[(t2,s)] + u[(color1, s, color2, t, slot)] -
                                2
```

```
for t1 in range(num_packages + 2):
    for t2 in range(num_packages + 2):
        for slot in range(1, num_slots + 1):
            for s in range(num_packages + 2):
                for t in range(num_packages + 2):
                    for color1 in needs_dict[s]:
                        for color2 in needs_dict[t]:
                            model += uo[(t1, t2, color1, s, color2, t, slot)] <= o[(
                                t1,s)]
                            model += uo[(t1, t2, color1, s, color2, t, slot)] <= o[(
                                t2,t)]
                            model += uo[(t1, t2, color1, s, color2, t, slot)] <= u
                                [(color1, s, color2, t, slot)]
                            model += uo[(t1, t2, color1, s, color2, t, slot)] >= o[(
                                t1,s)] + o[(t2,t)] + u[(color1, s, color2, t, slot)] -
                                2


# 保证0和最后时刻工序正常
model += o[(0,0)] == 1
model += o[(num_packages+1, num_packages+1)] == 1
#保证顺序有效性
for i in range(num_packages + 2):
    model += pl.lpSum(o[(i,j)] for j in range(num_packages + 2)) == 1
    model += pl.lpSum(o[(j,i)] for j in range(num_packages + 2)) == 1

# 保证每个颜色被转化进来
for t in range(1, num_packages + 1):
    for color2 in needs_dict[t]:
        model += pl.lpSum(ou[(t1,t2, color1, s, color2, t, slot)] \
                          for t1 in range(1, num_packages+1) \
                          for t2 in range(t1)\
                          for s in range(num_packages+2)\
                          for color1 in needs_dict[s]\
                          for slot in range(1, num_slots + 1)) == 1
```

```
# 保证每个颜色被转化出去
for s in range(1, num_packages + 1):
    for color1 in needs_dict[s]:
        model += pl.lpSum(uo[(t1,t2, color1, s, color2, t, slot)] \
                          for t1 in range(1, num_packages+1) \
                          for t2 in range(t1+1, num_packages+2)\
                          for t in range(num_packages+2)\
                          for color2 in needs_dict[t]\
                          for slot in range(1, num_slots + 1)) == 1

# 保证颜色转入转出是同一个槽
for slot in range(1, num_slots + 1):
    for time in range(1, num_packages + 1):
        for color in needs_dict[time]:
            model += pl.lpSum(ou[(t1,t2, color1, s, color, time, slot)] \
                          for t1 in range(num_packages+2) \
                          for t2 in range(t1)\
                          for s in range(num_packages+2)\
                          for color1 in needs_dict[s]) == \
                     pl.lpSum(uo[(t1,t2, color, time, color2, t, slot)] \
                          for t1 in range(num_packages+2) \
                          for t2 in range(t1+1, num_packages+2)\
                          for t in range(num_packages+2)\
                          for color2 in needs_dict[t])
# 保证顺序
for t in range(1, num_packages + 1):
    for color1, color2 in pairwise(needs_dict[t]):
        for max_slot in range(1, num_slots + 1):
            model += pl.lpSum(ou[(t1,t2, color3, s, color1, t, i)] \
                          for t1 in range(1, num_packages+1) \
                          for t2 in range(t1)\
                          for s in range(num_packages+2)\
                          for color3 in needs_dict[s]\
                          for i in range(1, max_slot + 1)) >= \
                     pl.lpSum(ou[(t1,t2, color3, s, color2, t, i)] \
                          for t1 in range(1, num_packages+1) \
                          for t2 in range(t1)\
```

```
                                for s in range(num_packages+2)\
                                for color3 in needs_dict[s]\
                                for i in range(1, max_slot + 1))
# 每条路从0转出1个
for slot in range(1, num_slots + 1):
    model += pl.lpSum(u[(0, 0, color2, t, slot)] for t in range(1, num_packages +
        2) for color2 in needs_dict[t]) == 1

# 每条路最后转化到1个
for slot in range(1, num_slots + 1):
    model += pl.lpSum(u[(color1, s, 0, num_packages + 1, slot)] for s in range(
        num_packages + 1) for color1 in needs_dict[s]) == 1

# 目标函数
model += pl.lpSum(time_table[color1 - 1][color2 - 1] * ou[(t1,t2, color1, s,
    color2, t, slot)]
                    for t1 in range(1,num_packages + 1)
                    for t2 in range(1,num_packages + 1)
                    for slot in range(1, num_slots + 1)
                    for s in range(1,num_packages + 1)
                    for t in range(1,num_packages + 1)
                    for color1 in needs_dict[s]
                    for color2 in needs_dict[t])
# 求解问题
model.solve(pl.GUROBI_CMD())

# 输出结果
for variable in model.variables():
    if variable.varValue > 0:
        print(f"{variable.name} = {variable.varValue}")

# 检查问题状态
print("Status:", pl.LpStatus[model.status])
```