2024年杭州电子科技大学数学建模竞赛

题　　目　基于整数规划的柔性印刷机墨盒切换问题研究

摘　　要：

柔性版印刷作为一种公认的绿色印刷方式，在瓦楞纸箱、标签等领域得到广泛应用并逐步占据主导地位。由于墨盒切换操作在柔性版印刷过程中占据了大量时间，因此优化印刷流程以减少印刷中的总切换时间对于优化柔性版印刷效率具有重要的研究意义。本文通过分析**柔性印刷机墨盒切换流程**中存在的各种约束条件，针对不同的问题建立了对应的**整数线性规划模型**，以合理配置每次印刷时插槽中的墨盒分布，从而显著提高印刷效率。

针对问题一，首先在单台清洗机并不考虑各包装中墨盒顺序的前提下，建立了满足**墨盒需求**和**插槽限制**等约束条件的**0-1整数线性规划模型**，以**最小化**墨盒的**总切换次数**。基于所建模型，使用了**Gurobi求解器**对附件一中的五个案例进行求解，最终求得五个案例的最小总切换次数分别为**5次、8次、48次、15次和130次**。

针对问题二，在问题一所建整数规划模型的基础上，调整其**辅助变量**与定义的**切换操作，**将目标函数修改为最小化总切换时间，建立了新的**整数线性规划模型，**然后，设计了**Gurobi方案**与**遗传算法方案**分别对附件二中的五个案例进行求解。经比较发现遗传算法求得的解相较Gurobi方案的解更优，且Gurobi方案无法求出案例4的最优解，因此案例4、5的最小总切换时间采用遗传算法方案的**近似最优解，**最终求得五个案例的最小总切换时间分别为**32min、51min、111min、345min和571min**。

针对问题三，在问题二模型的基础上新增**墨盒顺序**约束，从而得到修改后的**整数线性规划模型**，基于模型使用**Gurobi求解器**对附件三中的四个案例进行求解，最终求得四个案例的最小总切换时间分别为56**min、179min、263min和318min**。

针对问题四，在问题三的基础上需要在印刷之前确定**包装种类印刷顺序**，由此修改其决策变量以及相关的约束条件，并建立了新的**整数线性规划模型**，基于该模型，使用**Gurobi求解器**对附件四中的四个案例进行求解，最终求得四个案例的最小总切换时间分别为20**min、37min、66min和394min**。

针对问题五，在问题四的基础上，由单台清洗机变为两台清洗机，由此需要考虑切换过程中的**清洗机分配方案，**并根据两台清洗机的耗时确定最终的**最小化切换时间方案，**据此建立了修改后的**混合整数线性规划模型，**然后，基于该模型使用**Gurobi方案**与**遗传算法方案**对附件五中的四个案例进行求解，经比较发现遗传算法求得的解相较Gurobi方案的解更优，且Gurobi方案无法求出案例4的最优解，因此案例2、3、4的最小总切换时间采用了遗传算法方案的**近似最优解，**最终求得四个案例的最小总切换时间分别为16**min、165min、231min和277min**。

关键词：整数规划；线性规划；Gurobi求解器；遗传算法

目录

一、 问题重述

1.1 问题背景

柔性版印刷是印刷行业公认的一种绿色印刷方式，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方式，取代了传统的叠层铜的减材去除方式，具有减少浪费和快速原型设计等优势。

在一座印刷厂的大型柔性印刷机中，印刷过程包括将空白印刷材料放在印刷滚筒和印压滚之间，通过墨斗辊在墨盒中滚动并与网纹辊转动接触，再由网纹辊与印刷滚筒接触，将图案印刷到空白印刷材料上。该厂主要印刷食品类包装，如薯片包装、方便面包装、瓜子包装等。印刷机上有多个插槽，用于放置不同颜色的墨盒，插槽按照前后顺序排列，每个插槽对应一套传墨辊和滚筒。

由于插槽数量有限，无法一次性放置所有墨盒。在印刷某种包装时，插槽中的墨盒集合需包含该包装所需的颜色。印刷滚筒和印压滚之间的间隙大小可调，即使插槽中有其他不需要的墨盒存在，也不影响正常印刷。印刷完毕后，可能需要更换某个插槽中的墨盒。不同颜色墨盒的切换需要对传墨辊和滚筒进行清洗，花费时间称为切换时间。切换操作的频繁会增加印刷时间，影响印刷效率。因此减少总切换时间能够大幅提升印刷效率。

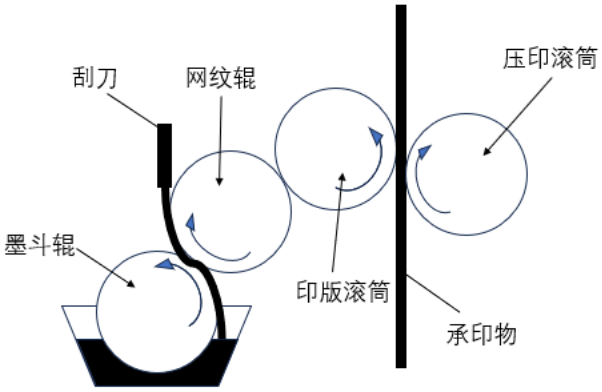

图 1 柔性印刷机工作原理示意图

1.2 问题提出

基于上述研究背景，题目提供了5个附件，分别包含了各包装种类所需墨盒编号、插槽序号、墨盒编号、包装种类编号、各包装种类所需墨盒编号等信息，本文需解决以下5个问题：

问题一：只有一台喷雾清洗机。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，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。在这种情况下，每种包装所需的墨盒顺序可以自由调整，且不考虑不同颜色墨盒切换时间的差别，因此优化的目标是最小化墨盒切换次数。

问题二：只有一台喷雾清洗机。不同颜色墨盒之间的切换时间不完全相同，且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，建立总切换时间最小化的数学模型，并根据附件2数据计算总切换时间。与问题一不同的是，不同颜色墨盒之间的切换时间不同，这要求考虑每次切换时所需的具

体时间，因此最终目标转变为最小化总切换时间。

问题三：只有一台喷雾清洗机。不同颜色墨盒之间的切换时间不完全相同，且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数据计算总切换时间。由于每种包装对应的墨盒顺序被固定了，因此需要增加新的约束条件来满足问题要求。

问题四：只有一台喷雾清洗机。不同颜色墨盒之间的切换时间不完全相同，且每种包装对应的墨盒顺序是不同且固定的。在印刷之前需要确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4数据计算总切换时间。在这一情况下，需要提前确定包装种类的印刷顺序，因此需要修改决策变量并调整约束条件来建立新的模型求解相关问题。

问题五：有两台喷雾清洗机。不同颜色墨盒之间的切换时间不完全相同，且每种包装对应的墨盒顺序是不同且固定的。在印刷之前需要确定包装种类印刷顺序的情况下，根据附件5数据计算总切换时间。本题与问题四类似但增加了一台清洗机从而可以并行处理切换操作，这需要增加新的辅助变量与约束条件来模拟两台清洗机的分配方案并附件5中的最小总切换时间。

二、 模型假设及符号说明

2.1 模型假设

1 假设进行墨盒切换时取出和放入墨盒的时间可以忽略，只需考虑清洗时间。
2 假设打印机第一次打印前插槽都为空且都是干净的。
3 假设打印机最后一次打印完成后不需要取出墨盒与清洗插槽。

2.2 符号说明

表2- 1 符号说明

符号	说明
M	待印刷包装集合，$\|M\| = m$
i	M中元素，即各包装
N	印刷机插槽集合，$\|N\| = n$
j	N中元素，即各插槽
S	墨盒集合，$\|S\| = s$
k,k'	S中元素，即各墨盒
P_i	包装i的需求墨盒集合,$\|P_i\| = n_{P_i}$
P_{il}	P_i中元素，即i所需墨盒,$1 \le l \le n_{p_i}$
$T_{kk'}$	由墨盒k切换到k'所需时间
Q	印刷次数集合，$\|Q\| = n_q = m$
q	Q中元素，即印刷次数
C	清洗机集合，$\|C\| = n_c$
c	C中元素，即清洗机c
Inf	无穷大的正数
φ	总切换次数
δ	总切换时间
x_{ijk}	0-1变量，表示包装i印刷完插槽j是否沾染墨盒k的颜色
y_{ij}	0-1变量，表示包装i印刷前插槽j是否切换墨盒
$y_{ijkk'}$	0-1变量，表示包装i印刷前插槽j中墨盒是否由k切换到k'
x_{qjk}	0-1变量，表示第q次印刷完插槽j是否沾染墨盒k的颜色
$y_{qjkk'}$	0-1变量，表示第q次印刷前插槽j中墨盒是否由k切换到k'
z_{qi}	0-1变量，表示第q次是否印刷包装i
$w_{qckk'}$	0-1变量，表示第q次印刷前清洗机c是否清洗墨盒k的颜色并换上k'
u_q	表示第q次印刷前清洗时长

三、 问题一模型建立与求解

3.1 问题分析

问题一在单台清洗机且不考虑每种包装中的墨盒顺序前提下根据给定的包装种类印刷顺序建立总切换次数最小化的数学模型，并根据附件1的数据计算总切换次数。通过仔细分析题目的要求，最后总结出以下几个核心问题：

1. 印刷顺序：需要按照给定的包装种类顺序进行包装印刷。
2. 墨盒需求：每种包装需要特定的墨盒集合，且墨盒数量与种类因包装种类而存在区别。
3. 插槽限制：每次包装印刷时插槽内的墨盒总数受到插槽总数量的限制,每个插槽中至多放置一个墨盒。
4. 切换操作：在当前插槽中的墨盒集合中缺少下一种包装所需要的集合时，需要取出当前插槽中不再需要的墨盒并换装所需要的墨盒，这一过程称为切换。清洗限制：插槽中每次切换不同的墨盒时需要对其对应的一套传墨辊和滚筒进行清洗。由于只有一台清洗机，因此同一时间只能清洗一套传墨辊和滚筒。
5. 主要目标：主要的目标是最小化墨盒总切换次数，从而提高印刷效率。

综合上述问题，需要建立一个综合考虑墨盒需求、插槽限制和切换操作的数学模型，主要目标是最小化总切换次数。由于在本问题中，墨盒是否放置在插槽中可以视为典型的二元决策问题，即放或不放，且需要满足多个线性约束条件，而整数规划方法方法能够有效处理这些离散变量和线性约束，并且在求解最小化问题方面有很强的理论支持，能够保证解的全局最优性，因此我们选择建立0-1整数线性规划模型来求解这一最小化总切换次数问题。

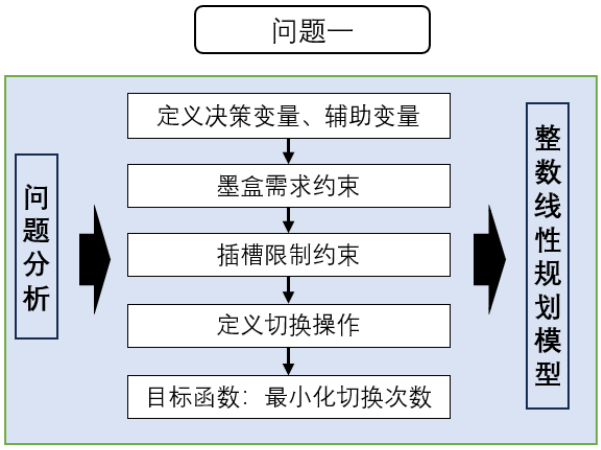

图 2问题一求解思路流程图

3.2 模型建立

3.2.1 决策变量

首先定义0-1型变量x_{ijk}表示包装i印刷完插槽j是否沾染墨盒k的颜色

$$x_{ijk}=\begin{cases}1，\ 包装i印刷完插槽j沾染墨盒k的颜色\\ 0，\ 否则\end{cases} \tag{3-1}$$

其次定义辅助0-1变量y_{ij}，表示包装i印刷前插槽j是否切换墨盒

$$y_{ij}=\begin{cases}1，\ 包装i印刷前插槽j切换墨盒\\0，\ 否则\end{cases} \tag{3-2}$$

3.2.2 目标函数

根据题目要求，我们最主要的优化思路是减少总切换时间来提高印刷效率，由于问题一中切换时间只与切换次数有关，因此建立的目标函数为最小化总切换次数：

$$\min\varphi = min\sum_{i=1}^{m}\sum_{j=1}^{n} y_{ij} \tag{3-3}$$

其中m表示问题中的包装种类数，n表示插槽数

3.2.3 约束条件

通过分析问题，我们所建立的约束条件如下：

1.墨盒需求：

每个包装的墨盒需要满足其需求：

$$\sum_{j=1}^{n} x_{ijk}=1 \quad \forall i\in M,\forall k\in P_i \tag{3-4}$$

其中P_i表示第i个包装印刷所需的墨盒编号集合。

2.插槽限制：

不同包装印刷结束时，每个插槽中至多沾染一个墨盒的颜色：

$$\sum_{k=1}^{s} x_{ijk}\le 1 \quad \forall i\in M,\forall j\in N \tag{3-5}$$

其中S表示墨盒颜色的总数。

插槽j沾上颜色后，不会恢复到干净状态

$$\sum_{k=1}^{s} x_{(i-1)jk}\le\sum_{k=1}^{s} x_{ijk} \quad \forall i\in M\backslash\{1\},\forall j\in N \tag{3-6}$$

3.定义切换操作：

当包装$i-1$印刷完和包装i印刷完插槽j都沾染有颜色且属于相同墨盒时，或不全沾染有颜色时，不需要进行切换操作，即$y_{ij}=0$，否则必须进行切换操作，即$y_{ij}=1$：

$$y_{ij}\ge x_{(i-1)jk}-x_{ijk} \quad \forall i\in M\backslash\{1\},\forall j\in N,\forall k\in S \tag{3-7}$$

3.2.4 最终模型

综上所述，建立如下关于最小化包装印刷墨盒总切换次数的整数线性规划模型：

$$\min\varphi = min\sum_{i=1}^{m}\sum_{j=1}^{n} y_{ij}$$

$$s.t.\begin{cases} x_{ijk} \in \{0,1\} \quad \forall i \in M, \forall j \in N, \forall k \in S \\ y_{ij} \in \{0,1\} \quad \forall i \in M, \forall j \in N, \forall k \in S \\ \sum_{j=1}^{n} x_{ijk} = 1 \quad \forall i \in M, \forall k \in P_i \\ \sum_{k=1}^{s} x_{ijk} \le 1 \quad \forall i \in M, \forall j \in N \\ \sum_{k=1}^{s} x_{(i-1)jk} \le \sum_{k=1}^{s} x_{ijk} \quad \forall i \in M\backslash\{1\}, \forall j \in N \\ y_{ij} \ge x_{(i-1)jk} - x_{ijk} \quad \forall i \in M\backslash\{1\}, \forall j \in N, \forall k \in S \end{cases}$$

3.3 问题求解

前文建立了最小化切换次数的整数线性规划模型，附件1给出了5个案例的数据，分别读取每个案例数据；使用Python语言的pulp包建立模型，并调用Gurobi求解器进行求解。

3.3.1 方案设计

1.求解步骤如下：

步骤1：获取案例数据

步骤2：使用pulp建立模型

步骤3：调用Gurobi求解器求解模型

步骤4：记录最小切换次数，并得到最优x_{ijk}和y_{ij}

步骤5：将x_{ijk}和y_{ij}转换为墨盒放置方案与切换方案，并输出

2.求解方案伪代码如下：

表3-1 案例一求解方案伪代码

输入：M,N,S,P
输出：最小切换次数，墨盒放置方案与切换方案
1. 使用pulp建立模型
2. 调用Gurobi求解器求解模型
3. 输出最小切换次数
4. 将x_{ijk}和y_{ij}转换为墨盒放置方案与切换方案
5. 输出墨盒放置方案与切换方案

其中x_{ijk}转换为墨盒放置方案思路为：

由于x_{ijk}表示包装i打印完插槽j中是否沾染墨盒k颜色，若只有一个插槽沾染了墨盒k的颜色时，包装i打印时墨盒k放置在该插槽内；若不止一个插槽沾染上墨盒k的颜色，则说明有一个插槽的墨迹来自之前的打印过程，此包装打印时墨盒k可以放置在任这些插槽的任意一个中，又考虑到之后的题目要求包装打印时要满足磨合顺序要求，因此将墨盒k放置到标号最小的插槽中。

由此也能看出墨盒放置的最优方案不唯一，但是最小切换次数是确定的。

3.3.2 结果展示

5个表格的最小切换次数汇总如下表：

表3-2问题一最小切换次数汇总表

案例	最小切换次数
Ins1_5_10_2	5

Ins2_7_10_3	8
Ins3_10_50_15	48
Ins4_20_50_10	15
Ins5_30_60_10	130

1. Ins1_5_10_2结果：

其最小切换次数为5次，包装对应墨盒放置方案如下：

表3- 3 Ins1_5_10_2墨盒放置方案

包装印刷顺序	插槽1	插槽2
1	5	7
2	5	2
3	4	2
4	4	8
5	3	7

2.Ins2_7_10_3结果：

其最小切换次数为8次，包装对应墨盒放置方案如下：

表3- 4 Ins2_7_10_3墨盒放置方案

包装印刷顺序	插槽1	插槽2	插槽3
1	7	2	
2	6	8	3
3	6	7	9
4	6	7	9
5	6	7	4
6	6	7	2
7	3	7	1

3. Ins3_10_50_15结果：

其最小切换次数为48次，包装对应墨盒放置方案如下：

表3- 5 Ins3_10_50_15墨盒放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
1	11									7					
2	11	17	10	6	42	24	4	8	31	23	35	40	33		21
3	11	15	10	29	16	30	4	8	31	23	27	1	43	14	19
4	12	15	10	44	16	30	4	48	31	23	27	1	43	14	19
5	12	26	10	44	16	30	4	48	31	38	27	41	43	6	19
6	5	26	10	44	16	30	4	48	31	42	27	25	43	6	19
7	47	26	17	22	16	30	4	48	31	7	27	25	43	28	13
8	23	26	49	19	16	18	4	6	36	7	27	15	40	28	33
9	34	26	12	8	47	18	4	6	20	3	1	39	41	28	5
10	34	26	24	9	47	18	4	6	20	3	38	39	41	2	5

4. Ins4_20_50_10结果：

其最小切换次数为15次，包装对应墨盒放置方案如下：

表3- 6 Ins4_20_50_10墨盒放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
1	25			30				11							28
2	25	2		30				11				36			28
3	25	2	1	30				11	41			36			28
4	25	2	1	30				11	41		12	36	22	13	28
5	25	2	1	30	46	40	7	11	37	31	12	36	22	13	28
6	25	2	1	30	46	40	7	11	37	31	12	36	22	38	28
7	25	2	34	30	46	15	7	11	14	31	12	36	35	38	28
8	44	2	34	30	46	15	27	11	14	31	26	3	35	21	28
9	44	2	42	20	46	18	27	11	14	31	26	4	35	21	28
10	44	2	42	20	46	18	27	11	14	31	26	4	35	21	28

5. Ins5_30_60_10结果：

其最小切换次数为130次，包装对应墨盒放置方案如下：

表3- 7 Ins5_30_60_10放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
1	56	5	11	55	26	44	14	9	33	20
2	56	10	11	55	26	44	14	4	33	49
3	56	10	3	55	26	38	14	4	33	49
4	56	35	3	54	2	19	36	4	46	8
5	56	35	3	24	33	19	36	34	10	8
6	56	11	3	24	48	19	36	40	10	12
7	56	11	3	24	48	19	36	40	10	12
8	20	14	18	1	5	51	30	40	37	12
9	20	11	18	38	5	32	30	40	37	10
10	43	44	18	38	26	57	50	56	16	10
11	22	44	6	38	52	57	50	56	16	10
12	22	44	46	38	52	57	50	24	16	10
13	22	44	46	28	52	57	50	59	16	10
14	22	3	30	1	15	49	34	37	20	10
15	22	23	30	1	15	49	19	37	36	10
16	28	7	14	41	15	33	38	18	58	10
17	3	29	14	41	15	11	38	18	8	10
18	3	29	56	46	15	31	48	50	8	10
19	45	29	56	46	26	31	48	42	8	10
20	57	20	56	54	51	24	48	40	37	10
21	57	20	56	54	35	24	48	40	37	10
22	16	25	56	54	23	24	48	40	37	10
23	43	25	14	12	49	24	9	40	3	10
24	32	25	4	22	33	24	46	40	3	13
25	32	25	4	22	30	21	46	40	3	23
26	54	16	35	41	30	48	17	50	3	29
27	54	16	59	41	30	48	17	50	3	34
28	51	49	59	41	38	47	10	36	3	6
29	16	42	59	41	44	13	10	36	3	1
30	16	30	59	41	9	13	33	39	3	24

四、 问题二模型建立与求解

4.1 问题分析

问题二要求在在单台清洗机且不考虑每种包装中的墨盒顺序的前提下根据给定的不同墨盒颜色之间的切换时间以及给定的包装种类印刷顺序来建立总结换时间最小化的数学模型，并根据附件2的数据计算总切换时间。

通过分析可以发现问题二与问题一的核心问题基本相同，但是由于考量了不同颜色墨盒之间切换时间的差异性，因此需要改变原有的辅助变量y_{ij}从而计算总切换时间。

由于该问题作为二元决策问题的性质没有改变，其核心问题也没有发生变化，因此本题选择在问题一的混合整数线性规划模型（MILP）基础上进行相应的改进来建立求解这一最小化总切换时间问题的数学模型。

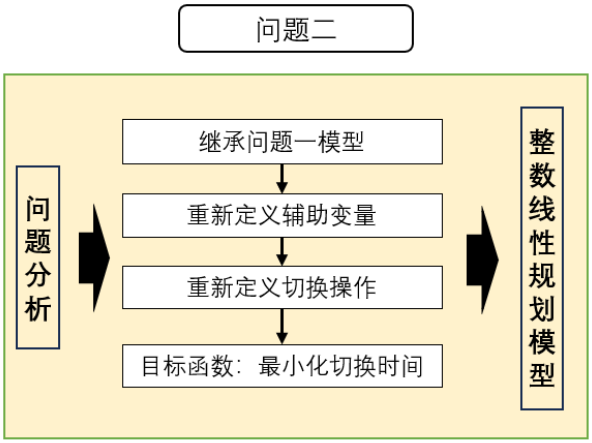

图 3问题二求解思路流程图

4.2 模型建立

4.2.1 决策变量

0-1型变量x_{ijk}同问题一相同，其辅助0-1变量$y_{ijkk'}$表示包装i印刷前插槽j中墨盒是否由k切换到k'：

$$y_{ijkk'} = \begin{cases} 1，\text{包装}i\text{印刷前插槽}j\text{中墨盒由}k\text{切换到}k' \\ 0，\text{否则} \end{cases} \quad (4-1)$$

4.2.2 目标函数

根据题目要求，我们的主要优化思路是最小化总切换时间来提高印刷效率，其对应的目标函数是最小化总切换时间：

$$\min \delta = min \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{s} \sum_{k'=1}^{s} T_{kk'} y_{ijkk'} \quad (4-2)$$

其中$T_{kk'}$表示由墨盒k切换到k'所需时间,即将墨盒k的颜色进行清洗并换入墨盒k'的切换时间。

4.2.3 约束条件

其墨盒需求与插槽限制的约束条件同问题一相同，而所定义的切换操作根据题目要求产生变化，变化后的切换操作如下所示：

1.定义切换操作：

当包装$i-1$印刷完和包装i印刷完插槽j中都存留有颜色则进行一次切换操作，否则不切换。

$$x_{(i-1)jk}+x_{ijk'}-1\leq y_{ijkk'} \quad \forall i\in M\backslash\{1\},\forall j\in N,\forall k,k'\in S \qquad (4-3)$$

注意：此时当两次留存颜色相同时，即$k=k'$,也算作进行切换操作，但是由于$T_{kk'}=0$;当$k=k'$;因此对于最终结果切换时间不会产生影响。

4.2.4 最终模型

综上所述，建立如下关于最小化包装印刷墨盒总切换时间的线性整数规划模型：

$$\min\delta=min\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{s}\sum_{j=1}^{s}T_{kk'}y_{ijkk'}$$

$$s.t.\begin{cases}x_{ijk}\in\{0,1\} \quad \forall i\in M,\forall j\in N,\forall k\in S\\ y_{ijkk'}\in\{0,1\} \quad \forall i\in M,\forall j\in N,\forall k,k'\in S\\ \sum_{j=1}^{n}x_{ijk}=1 \quad \forall i\in M,\forall k\in P_i\\ \sum_{k=1}^{s}x_{ijk}\leq 1 \quad \forall i\in M,\forall j\in N\\ \sum_{k=1}^{s}x_{(i-1)jk}\leq\sum_{k=1}^{s}x_{ijk} \quad \forall i\in M\backslash\{1\},\forall j\in N\\ x_{(i-1)jk}+x_{ijk'}-1\leq y_{ijkk'} \quad \forall i\in M\backslash\{1\},\forall j\in N,\forall k,k'\in S\end{cases}$$

4.3 问题求解

前文建立了最小化切换时间的整数线性规划模型，附件2给出了5个不同案例的数据，分别读取每个案例数据；使用Python语言的pulp包建立模型，并调用Gurobi求解器进行求解。其方案设计同问题一类似，新增输入变量T。

在求解中发现案例四经过较长时间后无法用求解器求出最优解，因此设计了遗传算法（GA）对问题二的各案例进行求解。

4.3.1 遗传算法方案设计

遗传算法是受自然进化理论启发的一系列搜索算法。通过模仿自然选择和繁殖的过程，遗传算法可以为涉及搜索，优化和学习的各种问题提供高质量的解决方案。同时，它们类似于自然进化，因此可以克服传统搜索和优化算法遇到的一些障碍，尤其是对于具有大量参数和复杂数学表示形式的问题。其算法流程图如下所示：

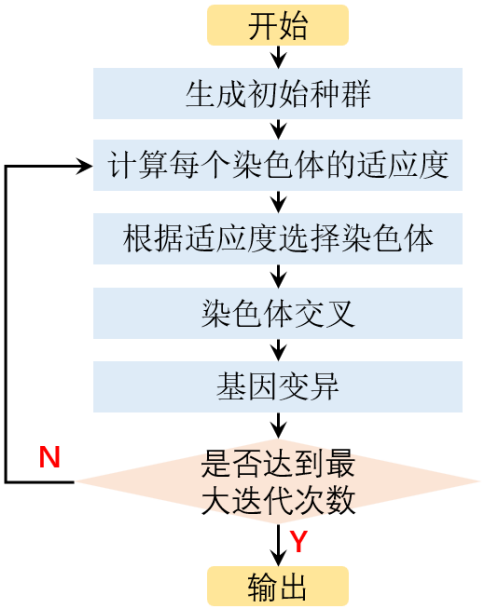

图 4 遗传算法流程图

1. 编码：

设定一个二维实数编码，行对应包装编号，列对应插槽编号，其实数编码x_{ij}代表包装i中插槽j染上的墨盒编号，0代表未染上颜色。矩阵示例如下：

$$\begin{bmatrix} 1 & 0 & 3 \\ 1 & 2 & 5 \\ 4 & 2 & 5 \end{bmatrix}$$

图 5 编码矩阵示例

其中$x_{23} = 5$表示包装2中插槽3上沾染了墨盒5的颜色。

2. 生成初始种群：

初始编码值默认为0，然后每次打印时将墨盒编号随机插入到插槽中，即对每一行的元素赋值，每次赋值时先继承上一行的编码，最终生成初始种群。

3. 计算适应度值：

染色体的适应度是指个体在种群生存的优势程度度量，适应度使用适应度函数来进行计算。适应度函数也叫评价函数，主要是通过个体特征从而判断个体的适应度，适应度函数是由目标函数变换而成。本文的适应度函数如下：

$$f(x) = \frac{1}{\delta + 1} \tag{4-4}$$

其中切换时间+1可以避免分母为0的情况发生。

4. 二元锦标赛选择：

锦标赛选择（Tournament Selection）是一种常用的个体选择方法，常用于遗传算法和进化计算中。它模拟了锦标赛的竞争过程，通过不断地选择优胜者来构建新一代的个体群体。其基本流程如下：

（1）从当前种群中随机选择若2个体作为参赛者。

（2）比较参赛者之间的适应度值，选择适应度最好的个体作为优胜者。

（3）重复以上步骤，直到选择出足够数量的优胜者。

5. 染色体交叉：

染色体交叉就是指在所选择的染色体中，选择两个进行相互交配，将它们的染色体按照某种方式相互交换部分基因，形成两个新的个体的过程。由于染色体各行间存在继承关系，因此本文中随机选择父类染色体的一列进行交叉互换，并得到子类染色体。

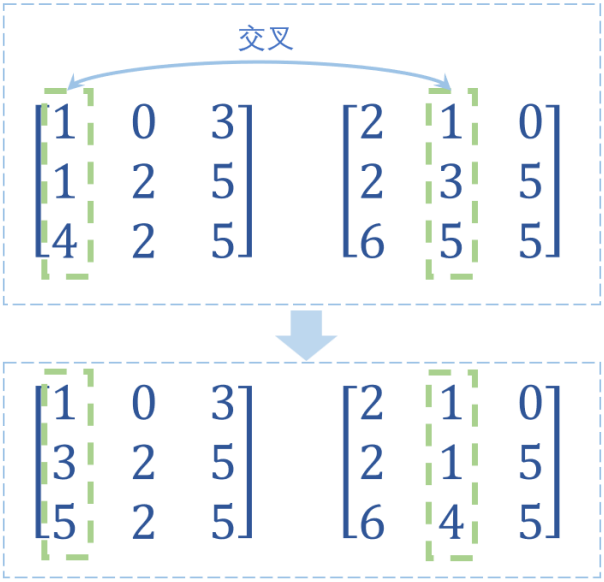

图 6 染色体交叉示例图

6. 基因变异：

在交叉操作过后形成的新个体，有一定的概率会发生基因变异，本文随机选择染色体编码中某一行中的两个基因进行交换。

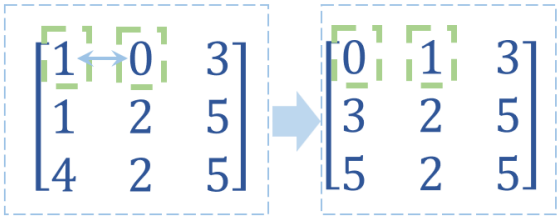

图 7 基因变异示例图

7. 冲突检测

（1） 重复颜色检测：检查并修正遗传算法中个体的第一行基因序列，以确保其中不存在重复的颜色。具体来说，它会找到所有重复的颜色，并将其中一个重复的颜色值设为零，最后返回修正后的个体。

（2） 交叉冲突检测：检查和修正遗传算法中个体的基因序列，确保在经过染色体交叉后每行基因序列包含规定中的所有元素。如果某个元素缺失，就将其添加到交叉列，最后返回修正后的个体。

4.3.2 参数设置

该遗传算法的部分参数设置如下：

参数名称	参数值
变异率	0.1
交叉率	0.8
种群大小	100

最大迭代次数	1000

4.3.3 结果展示

5个表格的最小切换时间汇总如下表：

表4-1 问题二最小切换时间汇总表

案例	最小切换时间（min）	
	Gurobi方案	遗传算法方案
Ins1_5_10_3	32	32
Ins2_7_10_2	43	51
Ins3_10_30_10	111	164
Ins4_20_40_10	-	345
Ins5_30_60_10	592	571

经比较可以发现遗传算法对案例5所求得的解比Gurobi方案更优，说明遗传算法方案在面对复杂度更高的问题时有更好的表现，而在复杂度较低，维度较少的案例2、3中Gurobi所求得的解相比遗传算法方案更优，表明Gurobi方案在面对低复杂度问题时其所求的解精度更高，更接近全局最优解。

1. Ins1_5_10_3结果：

其最小切换时间为32 min，包装对应墨盒放置方案如下：

表4-2 Ins1_5_10_3墨盒放置方案

包装印刷顺序	插槽1	插槽2	插槽3
1	3	6	4
2	3	6	5
3	9	1	5
4	9	1	5
5	9	1	2

2.Ins2_7_10_2结果：

其最小切换时间为43 min，包装对应墨盒放置方案如下：

表4-3 Ins2_7_10_2墨盒放置方案

包装印刷顺序	插槽1	插槽2
1		2
2	8	
3	5	3
4	4	3
5	5	6
6	3	1
7	6	1

3. Ins3_10_30_10结果：

其最小切换时间为111 min，包装对应墨盒放置方案如下：

表4-4 Ins3_10_30_10墨盒放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
1	3	22	12	10	11	8	21	16		19
2	11	23	12	19	14	26	20	18	2	5
3	11	25	12	29	14	8	20	24	1	21
4	11	25	4	29	27	8	20	10	1	21

5	9	7	4	29	27	8	17	10	1	22
6	9	7	23	29	27	8	4	20	1	22
7	9	7	23	29	27	8	4	20	1	22
8	9	7	23	3	27	8	4	20	1	22
9	9	18	23	3	27	14	4	20	24	22
10	9	18	16	3	27	14	12	20	24	22

4. Ins4_20_40_10结果：

其最小切换时间为345min，包装对应墨盒放置方案如下：

表4- 5 Ins4_20_40_10墨盒放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
1	2	9	28	24	15	6	4	20	29	25
2	31	9	39	19	36		4	20	18	26
3	31	20	39	30	36	9	7	8		26
4	31	21	14	36		9	34			26
5	1	21	5	36		9	34	19	27	17
6	12	29	28	10	14	23	34	25	27	17
7	2	36	1	24	12	23	32	26	9	17
8	2	22	1	24		23	32	21		11
9	12	19		5	13	34	20	26	10	11
10	12	5	30		13	35	20	26	10	1
11	33	16	23	15	19	30	20	18	5	1
12	33	16	18	15	19	30		32	6	1
13	38	10	21	15	26	3	19	24	6	36
14	4	31	20	39	26	33	21	11	6	1
15	23	31	12	33	26		21	11		1
16	29	30	36		33		21	11	9	37
17	39	32	36	13	26	29	24	8	20	27
18	39	31	36	19	26	11		8	1	22
19	39	31	36	24	26	11	19	8	34	22
20	39	31	2	24	26	11	20	8		33

5. Ins5_30_60_10结果：

其最小切换时间为571 min，包装对应墨盒放置方案如下：

表4- 6 Ins5_30_60_10放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
4			47	44	49				46	4
	7	18	47	32	0	52	28	56	46	
23	7	15	22	28	53	52	16	55	46	23
23	24	59	51	19	8	52	16	55	28	23
18	24	41	51	19	10	30	44	55	28	18
19	24	41	5	26	10	30	44	55	28	19
4	20	41	5	26	15	30		7	14	4
28	17	23	5	55		40	33	25	15	28
8	3	2	5	14	20	12	33	25	15	8
13	7	21	5	14	1	53	33	25	55	13
33	22	21	11	52	17	53		25	55	33
33	24	17	11	55	39	4	36	56	29	33
21	24	17	32	5	44	4	10	3	51	21
21	24	32	44		56	25	36	53	33	21
21	24	32	44	46	36	25		59	33	21
21	24	6	44	46	33	25	36			21
17	45	9	44	46	33	40	56	41	7	17

41	45	9	33	38		40	56	31	7	41
34	21	3	33	26	37	2	56	43	35	34
23	59	40	43		37	2	3	14	35	23
25	59	40	43		37	2	17	33	35	25
45	24	27	35	37	7	26	53	33	11	45
19	24	27	9	37	7	26	22	8	11	19
37	36	23	9	48	24	26	22	8		37
53	28	12	2	21	14	49	13	11	27	53
53	29	12	36	17	32	46	14	11	27	53
53	29	12	36	54	32	28	14	11	27	53
50	29	33	36	54	32	28	20	11	27	50
59	53	24	1	40	52	15	34	30	54	59
55	53	28	1	40	52	15	43	30	54	55

五、 问题三模型建立与求解

5.1 问题分析

问题三要求在单台清洗机的前提下根据给定的每种包装中的墨盒顺序、不同墨盒颜色之间的切换时间以及包装种类印刷顺序来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。

通过分析可以发现问题三在问题二的基础上考量了每种包装的墨盒顺序，因此需要新增墨盒印刷顺序的约束条件来计算总切换时间，由此本题计划在问题二的整数线性规划模型基础上增加新的约束来求解这一最小化总切换时间问题。

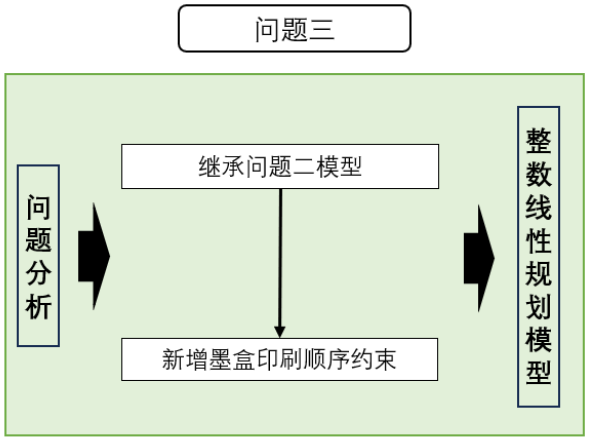

图 8 问题三求解思路流程图

5.2 模型建立

问题三的数学模型同问题二基本一致，只新增了一个约束条件即墨盒印刷顺序，其他决策变量与约束条件保持不变。

5.2.1 新增约束条件

1. 每种包装对应墨盒顺序：

   每种包装所需的墨盒必须按照表格所规定的顺序放入插槽中：

$$\sum_{j=1}^{j_1} x_{ijp_{il}} \leq \sum_{j=1}^{j_1} x_{ijp_{i(l-1)}} \quad \forall i \in M, \forall j_1 \in N, \forall l \in (1, n_{p_i}] \qquad (5-1)$$

其中p_{il}表示包装i中按序排列的墨盒编号集合，n_{p_i}表示包装i所需的墨盒种类数。

5.2.2 最终模型

综上所述，建立如下关于最小化总切换时间的线性整数规划模型：

$$\min \delta = min \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{s} \sum_{j=1}^{s} T_{kk'} \, y_{ijkk'}$$

$$
s.t.\begin{cases}
x_{ijk} \in \{0,1\} \quad \forall i \in M, \forall j \in N, \forall k \in S \\
y_{ijkk'} \in \{0,1\} \quad \forall i \in M, \forall j \in N, \forall k \in S, \forall k' \in S \\
\sum_{j=1}^{n} x_{ijk} = 1 \quad \forall i \in M, \forall k \in P_i \\
\sum_{k=1}^{s} x_{ijk} \le 1 \quad \forall i \in M, \forall j \in N \\
\sum_{k=1}^{s} x_{(i-1)jk} \le \sum_{k=1}^{s} x_{ijk} \quad \forall i \in M\backslash\{1\}, \forall j \in N \\
\sum_{j=1}^{j_1} x_{ijp_{il}} \le \sum_{j=1}^{j_1} x_{ijp_{i(l-1)}} \quad \forall i \in M, \forall j_1 \in N, \forall l \in (1, n_{p_i}] \\
\mathrm{x}_{(i-1)jk} + \mathrm{x}_{ijk'} - 1 \le y_{ijkk'} \quad \forall i \in M\backslash\{1\}, \forall j \in N, \forall k, k' \in S
\end{cases}
$$

5.3 问题求解

前文建立了最小化切换时间的整数线性规划模型，附件3给出了5个不同案例的数据，分别读取每个案例数据；使用Python语言的pulp包建立模型，并调用Gurobi求解器进行求解。其方案设计同问题一类似。

5.3.1 结果展示

4个表格的最小切换时间汇总如下表：

表5- 1问题三最小切换时间汇总表

表格	最小切换时间（min）
Ins1_5_5_2	56
Ins2_10_30_10	179
Ins3_20_50_10	263
Ins4_30_60_10	318

1. Ins1_5_5_2结果：

其最小切换时间为56min，包装对应墨盒放置方案如下：

表5- 2 Ins1_5_5_2墨盒放置方案

包装印刷顺序	插槽1	插槽2
1	3	2
2	2	1
3	1	3
4	3	1
5	2	3

2.Ins2_10_30_10结果：

其最小切换时间为179min，包装对应墨盒放置方案如下：

表5- 3 Ins2_10_30_10墨盒放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
1		29			23	5	25	8	28	22
2	22	29	6	17	26	15	9	12	23	7
3	22	29	6	17	27	15	11	12	8	7

4	11	29	6	17	16	15	11	12	8	7
5	17	11	6	22	29	15	2	21	26	7
6	24	11	26	22	9	8	5	21	26	7
7	26	5	23	22	9	3	24	15	4	7
8	20	29	26	22	17	18	4	2	13	7
9	7	29	6	23	17	18	25	28	22	24
10	7	29	19	23	17	18	25	28	22	24

3. Ins3_20_50_10结果：

其最小切换时间为263min，包装对应墨盒放置方案如下：

表5- 4 Ins3_20_50_10墨盒放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
1	3	4	21	37	26	29	39	14	32	35
2	30	36	33	49	8	45	42	25	32	9
3	1	25	33	7	8	30	35	25	36	17
4	20	25	41	7	8	36	35	25	14	17
5	20	25	36	7	8	36	17	43	14	4
6	35	25	25	7	8	43	4	5	14	6
7	35	25	23	7	8	43	28	5	14	38
8	17	5	21	7	22	43	6	3	12	10
9	6	31	21	14	46	43	6	26	34	38
10	6	8	21	16	46	43	6	26	45	10
11	33	8	18	16	46	43	6	24	3	10
12	6	8	25	7	46	43	23	25	40	28
13	29	8	19	7	18	1	21	23	40	26
14	29	8	21	17	19	1	38	23	9	12
15	18	15	21	47	19	1	25	23	4	12
16	25	35	33	47	12	37	14	23	15	24
17	10	35	33	40	12	37	20	23	45	24
18	14	35	33	40	34	28	10	23	12	24
19	14	35	49	29	45	28	10	43	8	24
20	14	35	49	29	45	16	10	1	41	24

4. Ins4_30_60_10结果：

其最小切换时间为318min，包装对应墨盒放置方案如下：

表5- 5 Ins4_30_60_10墨盒放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
1	31		15							
2		31		40	22	25	47	27	45	
3	14		27							
4		15		56	19	25	47		14	
5	4				19	25	47	20		
6		11	8		19	25	47	20	55	50
7	11		43		19	4	47		45	
8				47	19			55	45	41
9	37	35				43	48			
10	45	29	43	24	47	38	48		2	55
11	45	29	59		47	38	36	4	2	30
12	25	41	59	29	47	36	50		34	30
13	59	41	54	29	38	52	1	19	34	25
14	59	41	54	29	38	32	1	19	34	25
15	59	41	8	29	53	32	23	19	34	25
16	9	57	22	29	50	32	26	38	13	10
17	9	57	22	29	50	32	26	37	13	11

18	9	57	22	17	50	32	26	37	13	52
19	9	17	46	19	29	20	26	36	12	54
20	33	44	46	35	50	20	26	36	54	16
21	33	5	51	35	50	43	27	58	37	17
22	33	5	51	9	50	43	11	58	37	17
23	40	5	51	9	10	43	11	58	37	29
24	40	5	51	39	10	43	11	58	37	9
25	40	5	51	39	10	23	35	2	37	9
26	33	13	51	37	10	23	35	0	0	9
27	33	45	51	12	10	37	35	0	57	9
28	24	56	51	22	31	53	44	57	2	18
29	3	34	51	22	31	29	44	57	2	18
30	3	45	58	38	31	5	39	57	17	18

六、 问题四模型建立与求解

6.1 问题分析

问题四要求在单台清洗机的前提下根据给定的每种包装中的墨盒顺序、不同墨盒颜色之间的切换时间，同时在印刷之前需要确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4的数据计算总切换时间。

由于在问题四中需要在印刷前确定包装种类的印刷顺序，因此需要修改决策变量与相关的约束条件，并建立新的整数线性规划模型来求解这一最小化总切换时间问题。

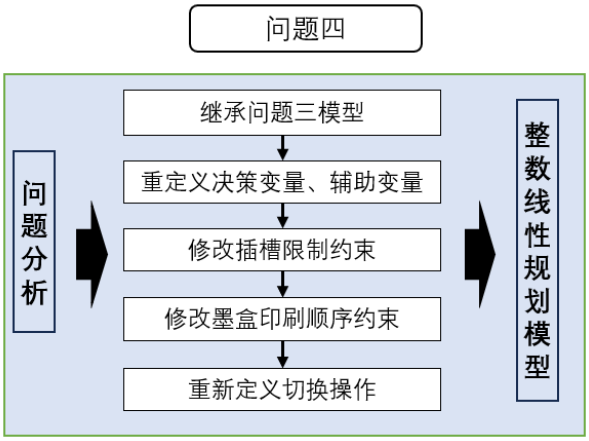

图 9 问题四求解思路流程图

6.2 模型建立

6.2.1 决策变量

首先定义0-1型变量x_{qjk}表示第q次印刷完插槽j是否沾染墨盒k的颜色

$$x_{qjk}=\begin{cases}1，\ 第q次印刷完插槽j沾染墨盒k的颜色\\0，\ 否则\end{cases} \tag{6-1}$$

辅助0-1变量$y_{ijkk'}$表示第q次印刷前插槽j中墨盒是否由k切换到k'

$$y_{qjkk'}=\begin{cases}1，\ 第q次印刷前插槽j中墨盒由k切换到k'\\0，\ 否则\end{cases} \tag{6-2}$$

新增辅助0-1变量z_{qi}，表示第q次是否印刷包装i：

$$z_{qi}=\begin{cases}1，\ 第q次印刷包装i\\0，\ 否\end{cases} \tag{6-3}$$

6.2.2 目标函数

根据题目要求，我们最主要的优化思路是减少总切换时间来提高印刷效率，因此建立的目标函数为最小化总切换时间：

$$min\,\delta=min\sum_{q=1}^{n_q}\sum_{j=1}^{n}\sum_{k=1}^{s}\sum_{k'=1}^{s}T_{kk'}y_{qjkk'} \tag{6-4}$$

其中Q表示总印刷次数。其值和包装种类总数M相等。

6.2.3 约束条件

通过分析问题，我们所建立的约束条件如下：

1. 墨盒需求：

   每次印刷包装的墨盒需要满足其需求：

$$\sum_{j=1}^{N} x_{qjk} \geq z_{qi} \quad \forall q \in Q, \forall k \in P_i, \forall i \in M \tag{6-5}$$

2. 插槽限制：

   每次印刷结束时，每个插槽中至多沾染一个墨盒的颜色：

$$\sum_{k=1}^{S} x_{qjk} \leq 1 \quad \forall q \in Q, \forall j \in N \tag{6-6}$$

   插槽沾染上颜色后只会改变不会消失：

$$\sum_{k=1}^{S} x_{(q-1)jk} \leq \sum_{k=1}^{S} x_{qjk} \quad \forall q \in Q\backslash\{1\}, \forall j \in N \tag{6-7}$$

3. 墨盒印刷顺序：

   每种包装所需的墨盒必须按照表格所规定的顺序放入插槽中：

$$\sum_{j=1}^{j_1} x_{qjp_{il}} - \sum_{j=1}^{j_1} x_{qjp_{i(l-1)}} \leq \left(1 - z_{qi}\right) \times Inf$$
$$\forall q \in Q, \forall i \in M, \forall j_1 \in N, \forall l \in (1, n_{p_i}] \tag{6-8}$$

   其中Inf表示一个无穷大的正数。`

4. 定义切换操作：

   当第$q-1$次印刷完和第q次印刷完插槽j中都存留有颜色则进行一次切换操作，否则不切换。

   注意：此时当两次留存颜色相同时，即$k = k'$,也算作进行切换操作，但是由于$T_{kk'} = 0$;当$k = k'$;因此对于最终结果切换时间不会产生影响。

$$x_{(q-1)jk} + x_{qjk'} - 1 \leq y_{qjkk'} \quad \forall q \in Q\backslash\{1\}, \forall j \in N, \forall k, k' \in S \tag{6-9}$$

5. 打印顺序约束

   每次打印必须打印一种包装

$$\sum_{i=1}^{m} z_{qi} = 1 \quad \forall q \in Q \tag{6-10}$$

   每种包装必须且仅打印一次

$$\sum_{q=1}^{n_q} z_{qi} = 1 \quad \forall i \in M \tag{6-11}$$

6.2.4 最终模型

综上所述，建立如下关于最小化包装印刷插槽墨盒总切换次数的线性整数规划模型：

$$\min \delta = min \sum_{q=1}^{n_q} \sum_{j=1}^{N} \sum_{k=1}^{S} \sum_{k'=1}^{S} T_{kk'} y_{qjkk'}$$

$$s.t. \begin{cases} x_{qjk} \in \{0,1\} \quad \forall q \in Q, \forall j \in N, \forall k \in S \\ y_{qjkk'} \in \{0,1\} \quad \forall q \in Q, \forall j \in N, \forall k \in S, \forall k' \in S \\ z_{qi} \in \{0,1\} \quad \forall q \in Q, \forall i \in M \\ \sum_{j=1}^{n} x_{qjk} \geq z_{qi} \quad \forall q \in Q, \forall k \in P_i, \forall i \in M \\ \sum_{k=1}^{s} x_{qjk} \leq 1 \quad \forall q \in Q, \forall j \in N \\ \sum_{j=1}^{j_1} x_{qjp_{il}} - \sum_{j=1}^{j_1} x_{qjp_{i(l-1)}} \leq \left(1 - z_{qi}\right) \times Inf \quad \forall q \in Q, \forall i \in M, \forall j_1 \in N, \forall l \in (1, n_{p_i}] \\ \mathrm{x}_{(q-1)jk} + \mathrm{x}_{qjk'} - 1 \leq y_{qjkk'} \quad \forall q \in Q \backslash \{1\}, \forall j \in N, \forall k, k' \in S \\ \sum_{i=1}^{m} z_{qi} = 1 \quad \forall q \in Q \\ \sum_{q=1}^{n_q} z_{qi} = 1 \quad \forall i \in M \end{cases}$$

6.3 问题求解

前文建立了最小化切换时间的整数线性规划模型，附件4给出了4个不同案例的数据，分别读取每个案例数据；使用Python语言的pulp包建立模型，并调用Gurobi求解器进行求解。其方案设计同问题二类似。

6.3.1 结果展示

4个表格的最小切换时间汇总如下表：

表6-1 问题四最小切换时间汇总表

案例	最小切换时间（min）
Ins1_5_10_2	20
Ins2_5_10_3	37
Ins3_7_10_2	65
Ins4_20_40_10	394

1. Ins1_5_10_2结果：

其最小切换时间为20min，包装对应墨盒放置方案如下：

表6-2 Ins1_5_10_2墨盒放置方案

包装印刷顺序	插槽1	插槽2
3	4	2
5	7	3

4	7	8
1	7	5
2	2	5

2. Ins2_5_10_3结果：

其最小切换时间为37min，包装对应墨盒放置方案如下：

表6- 3 Ins2_5_10_3墨盒放置方案

包装印刷顺序	插槽1	插槽2	插槽3
1	7	8	
4	7	6	4
3	3	6	4
5	3	1	6
2	4	1	5

3. Ins3_7_10_2结果：

其最小切换时间为65min，包装对应墨盒放置方案如下：

表6- 4 Ins3_7_10_2墨盒放置方案

包装印刷顺序	插槽1	插槽2
7	3	1
6	4	2
1	5	2
5	7	5
3	6	5
4	1	7
2	2	7

4. Ins4_20_40_10结果：

其最小切换时间为394min，包装对应墨盒放置方案如下：

表6- 5 Ins4_20_40_10墨盒放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
6	18	27	3							1
17	18	34	8	36		28	11		5	19
10	37	34	8	36		15	11		5	18
2	37	1	8	25		27			5	18
7	37	1	17	25	7	27	20		5	15
5	37	1	17		7	16	20	27	5	25
9	37	1	17	36	35	16	20	27	2	25
16	37	32	17	39	35	18	34	27	2	33
4	15	6	7	21	2	16	30	36	37	33
20	23	19	15	21	36	38	1	20	37	3
13	13	20	12	21	36	38	9		37	32
15	13	20	12	21	36	38	27		37	
14	28	20	12	37	36	38	27			21
3	38	20	37		4	18	27			21
18	38	20	37		4	18	29		27	21
11	33	20	37	29	4	18			27	21
8	33	20	29	26	17	21	19	7	35	10
12	33	20	8	26	5	21	19	37	3	10
19	4	12	38	20	22	9	32	25	28	16
1	23	12	38	27	22	9	32	25	28	16

七、 问题五模型建立与求解

7.1 问题分析

问题四要求在两台清洗机的前提下根据给定的每种包装中的墨盒顺序、不同墨盒颜色之间的切换时间，同时在印刷之前需要确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5的数据计算总切换时间。

通过分析可以发现问题五与问题四的核心问题基本相同，但是由于由单台清洗机变为两台清洗机，因此需要新增辅助变量$w_{qckk'}$,从而计算总切换时间。本题选择在问题一的整数线性规划模型基础上进行相应的改进来建立求解这一最小化总切换时间问题的数学模型。

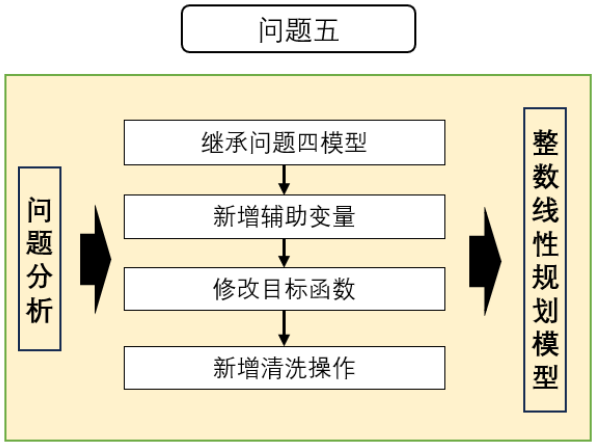

图 10 问题五求解思路流程图

7.2 模型建立

7.2.1 决策变量

首先新增辅助变量$w_{qckk'}$表示第q次印刷前清洗机c是否清洗墨盒k颜色并换上k'：

$$w_{qckk'}=\begin{cases}1，\ 第q次印刷前清洗机c清洗墨盒k颜色并换上k' \\ 0，\ 否则\end{cases} \tag{7-1}$$

其次新增辅助变量u_q，表示第q次印刷前切换操作总时间：

$$u_q \geq 0 \tag{7-2}$$

7.2.2 目标函数

根据题目要求，目标函数为最小化总切换时间：

$$min\,\delta = min\sum_{q=1}^{n_q} u_q \tag{7-3}$$

7.2.3 约束条件

其墨盒需求与插槽限制的约束条件同问题四相同，而所定义的切换操作根据题目要求产生变化，变化后的切换操作如下所示：

1. 清洗操作：

当发生切换操作时，就分配要分配清洗机进行清洗操作：

$$\sum_{c=1}^{n_c} w_{qckk'} \geq y_{qjkk'} \quad \forall q \in Q, \forall j \in N, \forall k, k' \in S \qquad (7-4)$$

其中n_c表示清洗机的台数，本题中$n_c = 2$。

2. 两台清洗机不同时进行同一个任务

$$\sum_{c=1}^{n_c} w_{qckk'} \leq 1 \quad \forall q \in Q, \forall k, k' \in S \qquad (7-5)$$

3. 每次清洗时长：

$$u_q \geq \sum_{q=1}^{n_q} \sum_{k=1}^{S} \sum_{k'=1}^{S} w_{qckk'} \times T_{kk'} \quad \forall c \in C \qquad (7-6)$$

7.2.4 最终模型

综上所述，建立如下关于最小化包装印刷墨盒总切换时间的整数线性规划模型：

$$\min \delta = min \sum_{q=1}^{n_q} u_q$$

$$s.t. \begin{cases} x_{qjk} \in \{0,1\} \quad \forall q \in Q, \forall j \in N, \forall k \in S \\ y_{qjkk'} \in \{0,1\} \quad \forall q \in Q, \forall j \in N, \forall k \in S, \forall k' \in S \\ z_{qi} \in \{0,1\} \quad \forall q \in Q, \forall i \in M \\ w_{qckk'} \in \{0,1\} \quad \forall q \in Q, \forall c \in C, \forall k, k' \in S \\ u_q \geq 0 \quad \forall q \in Q \\ \sum_{j=1}^{n} x_{qjk} \geq z_{qi} \quad \forall q \in Q, \forall k \in P_i, \forall i \in M \\ \sum_{k=1}^{S} x_{qjk} \leq 1 \quad \forall q \in Q, \forall j \in N \\ \sum_{j=1}^{j_1} x_{qjp_{il}} - \sum_{j=1}^{j_1} x_{qjp_{i(l-1)}} \leq (1 - z_{qi}) \times Inf \quad \forall q \in Q, \forall i \in M, \forall j_1 \in N, \forall l \in (1, n_{p_i}] \\ \mathrm{x}_{(q-1)jk} + \mathrm{x}_{qjk'} - 1 \leq y_{qjkk'} \quad \forall q \in Q \backslash \{1\}, \forall j \in N, \forall k, k' \in S \\ \sum_{i=1}^{m} z_{qi} = 1 \quad \forall q \in Q \\ \sum_{q=1}^{n_q} z_{qi} = 1 \quad \forall i \in M \\ \sum_{c=1}^{n_c} w_{qckk'} \geq y_{qjkk'} \quad \forall q \in Q, \forall j \in N, \forall k, k' \in S \\ \sum_{c=1}^{n_c} w_{qckk'} \leq 1 \quad \forall q \in Q, \forall k, k' \in S \\ u_q \geq \sum_{q=1}^{n_q} \sum_{k=1}^{S} \sum_{k'=1}^{S} w_{qckk'} \times T_{kk'} \quad \forall c \in C \end{cases}$$

7.3 问题求解

前文建立了最小化切换时间的整数线性规划模型，附件5给出了5个不同案例的数据，分别读取每个案例数据；使用Python语言的pulp包建立模型，并调用Gurobi求解器进行求解。其方案设计同问题二类似。

在求解中发现案例四经过较长时间后无法用求解器求出最优解，因此在改进问题二的遗传算法（GA）后使用该遗传算法方案对问题五的案例进行求解。

7.3.1 遗传算法改进设计

1. 改进适应度计算：

本题的适应度函数如下：

$$f(x) = \frac{1}{\delta + 1} \tag{4-4}$$

δ表示两台清洗机的清洗时间最大值，其中使用了贪婪算法对两台清洗机的分配方案进行求解。

2. 改进基因变异：

本题中为实现包装印刷顺序的变化，在基因变异中随机选择染色体编码中某两行中的所有基因进行交换。

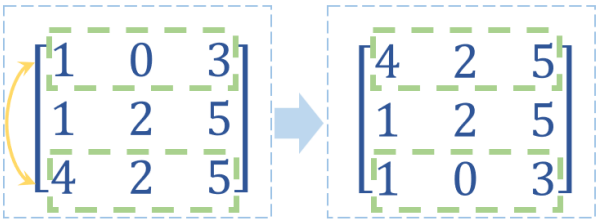

图 11 问题五基因变异示例图

3. 新增冲突检测：

（1） 顺序检测：根据规定的元素顺序确保每一行基因序列中的元素按照正确的顺序排列，如果发现顺序错误，则交换元素的位置以纠正顺序，最后返回调整后的个体。

（2） 变异检测：检查每列中的相邻行，如果发现某行的值为零而上一行的值不为零，则将上一行的值继承给当前行，从而确保颜色的连续性，最后返回修正后的染色体。

7.3.2 参数设置

该遗传算法的部分参数设置如下：

参数名称	参数值
变异率	0.3
交叉率	0.8
种群大小	100
最大迭代次数	1000

本题中增大变异率从而搜索到更多不同包装顺序的解。

7.3.3 结果展示

4个表格的最小切换时间汇总如下表：

表7-1 问题五最小切换时间汇总表

案例	最小切换时间（min）	
	Gurobi方案	遗传算法方案

Ins1_5_5_2	16	16
Ins2_10_50_15	168	165
Ins3_20_50_10	412	231
Ins4_30_60_10	-	277

经比较可以发现遗传算法方案在案例2、3上的解相较Gurobi方案更优，并且可以算出案例4的近似最优解，这可以说明遗传算法在面对复杂度更高、维度更多的整数规划问题时求解速度更快，所求得的解也更接近最优解。

1. Ins1_5_5_2结果：

其最小切换时间为16min，包装对应墨盒放置方案如下：

表7- 2 Ins1_5_5_2墨盒放置方案

包装印刷顺序	插槽1	插槽2
1	4	1
2	2	1
5	2	4
4	3	2
3	4	3

其清洗机清洗分配方案如下：

表7- 3 Ins1_5_5_2清洗机分配方案

印刷次数序号	清洗机编号	插槽序号	墨迹所属墨盒	新放置墨盒编号
2	1	1	4	2
3	2	2	1	4
4	1	1	2	3
4	2	2	4	2
5	1	1	3	4

2. Ins2_10_50_15结果：

其最小切换时间为165min，包装对应墨盒放置方案如下：

表7- 4 Ins2_10_50_15墨盒放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
6	0	5	42	0	0	25	0	0	0	0	0	0	0	0	0
2	31	5	6	0	40	21	10	17	35	42	0	23	0	24	33
4	31	23	19	30	40	21	48	17	35	42	12	10	44	0	33
10	31	23	19	30	40	2	48	17	24	38	6	39	9	0	4
3	11	23	16	30	27	1	10	19	8	29	14	0	43	15	4
5	4	23	16	30	38	41	10	0	8	29	26	0	43	15	6
8	4	40	16	36	33	19	18	23	26	6	15	27	49	0	7
9	26	8	20	41	6	5	18	3	4	39	47	12	1	34	28
7	25	27	13	41	6	31	30	22	43	28	16	48	7	47	17
1	25	27	13	11	6	31	30	22	43	28	16	48	7	47	0

3. Ins3_20_50_10结果：

其最小切换时间为231min，包装对应墨盒放置方案如下：

表7- 5 Ins3_20_50_10墨盒放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
7		43		30						
20	28	49		30					25	13
15	16	49		12	41	15		4	31	28

16	31	21		7	2	35	24	4	32	1
1	19	21	49	7	22	35	41	0	32	39
2	28	8	22	4	48	7	19	41	9	33
4	35	8	6	46	48	7	4	41	9	28
14	35	8	6	43	41	23		17	46	28
11	35	8	6	43	41	23		17	5	28
18	35	38	6	43	41			34	5	28
19	42	48	20	38	2	23	40	4	22	37
9	6	27	34	48	36	26	12	29	20	37
8	23	27	38	15	36	26	31	29	20	37
10	23	40	39	4	36	3	31	22	27	35
3	15	40	18	49	12	3	20	35	29	25
6	20	37	38	33	11	22	10	32	25	43
12	20	37	38	33	21	13	10	1	41	43
17	20	40	10	49	45	13	46	1	31	43
5	20	21	10	49	45	23	26	1	14	43
13	3	21		49	45	23	26	11	14	9

4. Ins4_30_60_10结果：

其最小切换时间为277min，包装对应墨盒放置方案如下：

表7- 6 Ins4_30_60_10墨盒放置方案

m	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
14		18	34	57	23			7	56	6
4	12	50	9	43	21	33	53	40	15	31
10	27	43	52	30	11	9	1	17	21	38
23	29	43	23	35	12	44	1	20	21	17
12	29	36	23	35	16	25	51	13	26	17
8	29	14	22	49	16	25	51	13	9	13
28	29	55	15	49	51	25		13	9	13
24	29	43	17	49	51	25	19	24	9	13
6	29	43	17	55	11	25	20	48	7	13
30	43	43	17	55	11	25	22	48	7	
22	34	43	17	55	17	25	22	48	59	
15	34	43	17	53	17	25	22	48	59	44
2	38	59	34	2	3	19	52	55	10	40
18	38		34	2	39	19	52	55	10	40
27	49	26	34	2	39	19	52	55	10	40
16	49	16	34	2	39	19	52	55	24	10
3	49	54	34	2	39	19	52	55	24	10
29	59	54	34	2	39	19	12	55	14	10
7	59	18	5	25	39	13	38	4	14	10
17	59	18	5	25	39	13	38	4	26	33
1	23	18	49	12	5	32	38	43	51	33
9	23	18	49	27	5	32	38	43	58	33
26	23	18	47	35	5	34	38	43	49	33
13	23	55	52	35	5	50	1	7	49	27
11	41	7	52	2	5	50	42		21	27
19	41	7	52	2	25	50	6		21	27
5	7	13	45	19	25	48	40	31	36	57
21	36	40	45	17	56	10	25	31	24	57
20	36	46	45	17	56	10	20	31	2	21
25	17	21	32	14	10	6	30	31	24	27

八、 模型评价

8.1 模型的优点

（1）决策最优化：0-1整数线性规划模型可以在优化过程中精确地控制变量取值，确保每个决策的最优性。这种方法能够有效地最小化切换时间，从而提高印刷效率。

（2）模型适应性：模型能够处理不同颜色墨盒之间切换时间不同的复杂情况。通过引入不同的决策变量，模型可以灵活地适应不同的印刷需求。

（3）约束条件完备：模型中包含了多个约束条件，确保了墨盒切换过程的合理性和可行性，保证了模型的实际应用价值。

8.2 模型的缺点

（1）计算复杂度高：0-1整数线性规划模型在变量和约束数量较多时，计算复杂度会显著增加，导致求解时间过长，尤其在印刷厂规模较大、墨盒数量较多的情况下。

（2）静态优化：当前模型假设在印刷过程中每个包装的墨盒顺序和印刷顺序在优化前已经确定，缺乏动态调整和实时优化的能力。在实际生产过程中，可能需要根据生产情况动态调整，这方面的灵活性有待提高。

8.3 模型的推广与改进

（1）动态优化模型：引入动态优化或实时调整机制，使模型能够根据实际生产情况和即时数据进行调整，从而进一步提高印刷效率和适应性。

（2）多目标优化：在考虑最小化切换时间的同时，可以引入其他优化目标，如最小化墨盒消耗等，通过多目标优化模型，综合考虑多个因素，提高模型的实际应用价值。

九、 参考文献

[1] Holland J H. Genetic algorithms[J]. Scientific american, 1992, 267(1): 66-73.
[2] 清华大学《运筹学》教材编写组.运筹学(第三版)[M]清华大学出版社,2005.
[3] CSDN, 数学建模线性规划（整数规划）, https://blog.csdn.net/qq_52136639/article/details/127496551, 2024.6.1.
[4] CSDN, python通过Gurobi求解线性规划, https://blog.csdn.net/weixin_42828342/article/details/138293605, 2024.6.1.

十、附录

代码1

介绍：用Python设计Gurobi方案对第1题求解

```
import pulp
import numpy as np
import pandas as pd
import re
import openpyxl
def cal_Q1(M,N,S,P):
    # 创建问题
    prob = pulp.LpProblem("Minimize_Switching", pulp.LpMinimize)
    # 决策变量：x_ijk
    x = pulp.LpVariable.dicts("x", ((i, j, k) for i in range(N) for j in
range(S) for k in range(1, M+1)), 0, 1, pulp.LpBinary)
    # 辅助变量：y_ij 表示在第 i 个包装的第 j 个插槽中是否进行切换
    y = pulp.LpVariable.dicts("y", ((i, j) for i in range(1, N) for j in
range(S)), 0, 1, pulp.LpBinary)
    # 目标函数：最小化总切换次数
    prob += pulp.lpSum(y[i, j] for i in range(1, N) for j in range(S))
    # 约束条件
    for i in range(N):
        for j in range(S):
            # 每个包装的每个插槽只能放一个墨盒
            prob += pulp.lpSum(x[i, j, k] for k in range(1, M+1)) <= 1
    for i in range(N):
            # 每个包装的墨盒集合必须满足其需求
            for k in P[i]:
                prob += pulp.lpSum(x[i, j, k] for j in range(S)) == 1
    for i in range(N):
        for j in range(S):
            # 每个墨盒每次只能放在一个插槽
            prob += pulp.lpSum(x[i,j,k] for k in range(1,M+1)) <= 1
    for i in range(1, N):
        for j in range(S):
            prob += pulp.lpSum(x[i-1,j,k] for k in range(1,M+1)) <=
pulp.lpSum(x[i,j,k] for k in range(1,M+1))
    # 定义切换操作
    for i in range(1, N):
        for j in range(S):
            for k in range(1, M+1):
                prob += x[i-1, j, k] - x[i,j,k] <= y[i, j]
    # 求解问题
    solver = pulp.GUROBI_CMD(timeLimit=7200,threads=12,msg=0)
    prob.solve(solver)
    # 输出结果
    print("Status:", pulp.LpStatus[prob.status])
    print("最少切换次数:", pulp.value(prob.objective))
```

```
    project = np.zeros((N,S))
    switch = np.zeros((N,S))
    for i in range(N):
        for j in range(S):
            for k in range(1, M+1):
                if pulp.value(x[i, j, k]) == 1:
                    project[i,j] = k
                    # print(f"包装 {i+1}, 插槽 {j+1} -> 墨盒 {k}")
                    break
    print('--------------------------------------------------')
    # 输出 y
    for i in range(1,N):
        for j in range(S):
            if pulp.value(y[i, j]) == 1:
                switch[i,j] = 1
                # print(f"包装 {i+1}, 插槽 {j+1} -> 更换")
    return project,switch
def read_data(file_path,N,M,S):
    path = 'D:\\Git 仓库\\machinelearning_ZZLclass\\0531\\附件数据（B 题）\\附件
1\\' + file_path
    data_1 = pd.read_excel(path, sheet_name='包装种类及其所需墨盒')
    print_info = []
    for i in range(N):
        num = re.findall(r'\d+', data_1.iloc[i, 1])
        print_info.append([int(j) for j in num])
    data_1 = print_info
        return data_1
# 附件 1 数据
print('案例 1 求解')
N = 5  # 包装种类数量
M = 10  # 墨盒数量
S = 2  # 插槽数量
# 包装所需的墨盒集合
P = read_data('Ins1_5_10_2.xlsx',N,M,S)
project1,switch1 = cal_Q1(M,N,S,P)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project1_1.csv',project1,delimiter=',',fmt='%d')
np.savetxt('switch1_1.csv',switch1,delimiter=',',fmt='%d')
# 附件 2 数据
print('案例 2 求解')
N = 7  # 包装种类数量
M = 10  # 墨盒数量
S = 3  # 插槽数量
# 包装所需的墨盒集合
P = read_data('Ins2_7_10_3.xlsx',N,M,S)
project2,switchcar2 = cal_Q1(M,N,S,P)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project1_2.csv',project2,delimiter=',',fmt='%d')
```

```
np.savetxt('switch1_2.csv',switchcar2,delimiter=',',fmt='%d')
# 附件 3 数据
print('案例 3 求解')
N = 10  # 包装种类数量
M = 50  # 墨盒数量
S = 15  # 插槽数量
# 包装所需的墨盒集合
P = read_data('Ins3_10_50_15.xlsx',N,M,S)
project3,switchcar3 = cal_Q1(M,N,S,P)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project1_3.csv',project3,delimiter=',',fmt='%d')
np.savetxt('switch1_3.csv',switchcar3,delimiter=',',fmt='%d')
# 附件 4 数据
print('案例 4 求解')
N = 10  # 包装种类数量
M = 50  # 墨盒数量
S = 15  # 插槽数量
# 包装所需的墨盒集合
P = read_data('Ins4_10_50_15.xlsx',N,M,S)
project4,switchcar4 = cal_Q1(M,N,S,P)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project1_4.csv',project4,delimiter=',',fmt='%d')
np.savetxt('switch1_4.csv',switchcar4,delimiter=',',fmt='%d')
# 附件 5 数据
print('案例 5 求解')
N = 30  # 包装种类数量
M = 60  # 墨盒数量
S = 10  # 插槽数量
# 包装所需的墨盒集合
P = read_data('Ins5_30_60_10.xlsx',N,M,S)
project5,switchcar5 = cal_Q1(M,N,S,P)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project1_5.csv',project5,delimiter=',',fmt='%d')
np.savetxt('switch1_5.csv',switchcar5,delimiter=',',fmt='%d')
```

代码2

介绍：用Python设计Gurobi方案对第2题求解

```
import pulp
import numpy as np
import pandas as pd
import re
import openpyxl
def cal_Q2(N,M,S,P,T):
    # 创建问题
    prob = pulp.LpProblem("Minimize_Switching_Time", pulp.LpMinimize)
    # 决策变量：x_ijk
    x = pulp.LpVariable.dicts("x", ((i, j, k) for i in range(N) for j in
range(S) for k in range(1, M+1)), 0, 1, pulp.LpBinary)
    # 辅助变量：y_ijkk'
```

```
    y = pulp.LpVariable.dicts("y", ((i, j, k_prime, k) for i in range(1, N)
for j in range(S) for k_prime in range(1, M+1) for k in range(1, M+1)), 0, 1,
pulp.LpBinary)
    # 目标函数：最小化总切换时间
    prob += pulp.lpSum(T[k_prime-1][k-1] * y[i, j, k_prime, k] for i in
range(1, N) for j in range(S) for k_prime in range(1, M+1) for k in range(1,
M+1))
    # 约束条件
    for i in range(N):
        for j in range(S):
            # 每个包装的每个插槽只能放一个墨盒
            prob += pulp.lpSum(x[i, j, k] for k in range(1, M+1)) <= 1
    for i in range(N):
            # 每个包装的墨盒集合必须满足其需求
            for k in P[i]:
                prob += pulp.lpSum(x[i, j, k] for j in range(S)) == 1
    for i in range(1, N):
        for j in range(S):
            prob += pulp.lpSum(x[i-1,j,k] for k in range(1,M+1)) <=
pulp.lpSum(x[i,j,k] for k in range(1,M+1))
    # 定义切换操作
    for i in range(1, N):
        for j in range(S):
            for k_prime in range(1, M+1):
                for k in range(1, M+1):
                    prob += y[i, j, k_prime, k] >= x[i-1, j, k_prime] + x[i, j,
k] - 1
    # 求解问题
    solver = pulp.GUROBI_CMD(timeLimit=7200,threads=12,msg=0)
    prob.solve(solver)
    # 输出结果
    print("Status:", pulp.LpStatus[prob.status])
    print("最少切换时间:", pulp.value(prob.objective))
    project = np.zeros((N,S))
    switch = []
    for i in range(N):
        for j in range(S):
            for k in range(1, M+1):
                if pulp.value(x[i, j, k]) == 1:
                    project[i,j] = k
                    # print(f"包装 {i+1}, 插槽 {j+1} -> 墨盒 {k}")
                    break
    print('-----------------------------------------------------')
    # 输出 y
    for i in range(1,N):
        for j in range(S):
            for k_prime in range(1,M+1):
                for k in P[i]:
```

```
                    if pulp.value(y[i, j,k_prime,k]) == 1:
                        if k_prime != k:
                            switch.append([i+1,j+1,k_prime,k])
                            # print(f"包装 {i+1}, 插槽 {j+1} ->从墨盒{k_prime} 更
换为{k}")
    switch = np.array(switch)
    return project,switch
def read_data(file_path,N,M,S):
    path = 'D:\\Git 仓库\\machinelearning_ZZLclass\\0531\\附件数据（B 题）\\附件
2\\' + file_path
    data_1 = pd.read_excel(path, sheet_name='包装种类及其所需墨盒')
    data_2 = pd.read_excel(path, sheet_name='墨盒切换时间')
    print_info = []
    for i in range(N):
        num = re.findall(r'\d+', data_1.iloc[i, 1])
        print_info.append([int(j) for j in num])
    data_1 = print_info
    data_2 = data_2.values[:,1:]
    return data_1,data_2
# 附件 1 数据
print('案例 1 求解')
N = 5  # 包装种类数量
M = 10  # 墨盒数量
S = 3  # 插槽数量
# 获取 PT 数据
P,T = read_data('Ins1_5_10_3.xlsx',N,M,S)
project2_1,switch2_1 = cal_Q2(N,M,S,P,T)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project2_1.csv',project2_1,delimiter=',',fmt='%d')
np.savetxt('switch2_1.csv',switch2_1,delimiter=',',fmt='%d')
# 附件 2 数据
print('案例 2 求解')
N = 7  # 包装种类数量
M = 10  # 墨盒数量
S = 2  # 插槽数量
P,T = read_data('Ins2_7_10_2.xlsx',N,M,S)
project2_2,switch2_2 = cal_Q2(N,M,S,P,T)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project2_2.csv',project2_2,delimiter=',',fmt='%d')
np.savetxt('switch2_2.csv',switch2_2,delimiter=',',fmt='%d')
# 附件 3 数据
print('案例 3 求解')
N = 10  # 包装种类数量
M = 30  # 墨盒数量
S = 10  # 插槽数量
P,T = read_data('Ins3_10_30_10.xlsx',N,M,S)
project2_3,switch2_3 = cal_Q2(N,M,S,P,T)
# 把 project 和 switchcar 存为 csv 文件
```

```
np.savetxt('project2_3.csv',project2_3,delimiter=',',fmt='%d')
np.savetxt('switch2_3.csv',switch2_3,delimiter=',',fmt='%d')
# 附件 4 数据
print('案例 4 求解')
N = 20  # 包装种类数量
M = 40  # 墨盒数量
S = 10  # 插槽数量
P,T = read_data('Ins4_20_40_10.xlsx',N,M,S)
project2_4,switch2_4 = cal_Q2(N,M,S,P,T)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project2_4.csv',project2_4,delimiter=',',fmt='%d')
np.savetxt('switch2_4.csv',switch2_4,delimiter=',',fmt='%d')
# 附件 5 数据
print('案例 5 求解')
N = 30  # 包装种类数量
M = 60  # 墨盒数量
S = 10  # 插槽数量
P,T = read_data('Ins5_30_60_10.xlsx',N,M,S)
project2_5,switch2_5 = cal_Q2(N,M,S,P,T)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project2_5.csv',project2_5,delimiter=',',fmt='%d')
np.savetxt('switch2_5.csv',switch2_5,delimiter=',',fmt='%d')
```

代码3

介绍：用Python设计Gurobi方案对第3题求解

```
import pulp
import numpy as np
import pandas as pd
import re
import openpyxl
def cal_Q3(N,M,S,P,T,N_p):
    # 创建问题
    prob = pulp.LpProblem("Minimize_Switching_Time", pulp.LpMinimize)
    # 决策变量：x_ijk
    x = pulp.LpVariable.dicts("x", ((i, j, k) for i in range(N) for j in
range(S) for k in range(1, M+1)), 0, 1, pulp.LpBinary)
    # 辅助变量：y_ijkk'
    y = pulp.LpVariable.dicts("y", ((i, j, k_prime, k) for i in range(1, N)
for j in range(S) for k_prime in range(1, M+1) for k in range(1, M+1)), 0, 1,
pulp.LpBinary)
    # 目标函数：最小化总切换时间
    prob += pulp.lpSum(T[k_prime-1][k-1] * y[i, j, k_prime, k] for i in
range(1, N) for j in range(S) for k_prime in range(1, M+1) for k in range(1,
M+1))
    # 约束条件
    for i in range(N):
        for j in range(S):
            # 每个包装的每个插槽只能放一个墨盒
            prob += pulp.lpSum(x[i, j, k] for k in range(1, M+1)) <=
```

```
    for i in range(N):
            # 每个包装的墨盒集合必须满足其需求
            for k in P[i]:
                prob += pulp.lpSum(x[i, j, k] for j in range(S)) == 1
    # 插槽沾染颜色后便一直会沾染一种颜色
    for i in range(1, N):
        for j in range(S):
            prob += pulp.lpSum(x[i-1,j,k] for k in range(1,M+1)) <=
pulp.lpSum(x[i,j,k] for k in range(1,M+1))
    # 定义切换操作
    for i in range(1, N):
        for j in range(S):
            for k_prime in range(1, M+1):
                for k in range(1, M+1):
                    prob += y[i, j, k_prime, k] >= x[i-1, j, k_prime] + x[i, j,
k] - 1
    # 满足墨盒顺序
    for i in range(N):
        for j1 in range(S):
                for l in range(1,N_p[i]):
                    prob += pulp.lpSum(x[i,j,P[i][l-1]] for j in range(j1)) >=
pulp.lpSum(x[i,j,P[i][l]] for j in range(j1))
    # 求解问题
    solver = pulp.GUROBI_CMD(timeLimit=7200,threads=12,msg=0)
    prob.solve(solver
    # 输出结果
    print("Status:", pulp.LpStatus[prob.status])
    print("最少切换时间:", pulp.value(prob.objective)
    project = np.zeros((N,S))
    switch = []
    for i in range(N):
        for j in range(S):
            for k in range(1, M+1):
                if pulp.value(x[i, j, k]) == 1:
                    project[i,j] = k
                    # print(f"包装 {i+1}, 插槽 {j+1} -> 墨盒 {k}")
                    break
    print('-----------------------------------------------------')
    # 输出 y
    for i in range(1,N):
        for j in range(S):
            for k_prime in range(1,M+1):
                for k in range(1,M+1):
                    if pulp.value(y[i, j,k_prime,k]) == 1:
                        if k_prime != k:
                            switch.append([i+1,j+1,k_prime,k])
                            # print(f"包装 {i+1}, 插槽 {j+1} ->从墨盒{k_prime} 更
换为{k}")
```

```
    switch = np.array(switch)
    return project,switch
def read_data(file_path,N,M,S):
    path = 'D:\\Git 仓库\\machinelearning_ZZLclass\\0531\\附件数据（B 题）\\附件
3\\' + file_path
    data_1 = pd.read_excel(path, sheet_name='包装种类及其所需墨盒')
    data_2 = pd.read_excel(path, sheet_name='墨盒切换时间')
    print_info = []
    for i in range(N):
        num = re.findall(r'\d+', data_1.iloc[i, 1])
        print_info.append([int(j) for j in num])
    data_1 = print_info
    N_p = [len(data_1[i]) for i in range(N)]
    data_2 = data_2.values[:,1:]
    return data_1,data_2,N_p
# 附件 1 数据
print('案例 1 求解')
N = 5  # 包装种类数量
M = 5  # 墨盒数量
S = 2  # 插槽数量
# 获取 PT 数据
P,T,N_p = read_data('Ins1_5_5_2.xlsx',N,M,S)
project3_1,switch3_1 = cal_Q3(N,M,S,P,T,N_p)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project3_1.csv',project3_1,delimiter=',',fmt='%d')
np.savetxt('switch3_1.csv',switch3_1,delimiter=',',fmt='%d')
# 附件 2 数据
print('案例 2 求解')
N = 10  # 包装种类数量
M = 30  # 墨盒数量
S = 10 # 插槽数
# 获取 PT 数据
P,T,N_p = read_data('Ins2_10_30_10.xlsx',N,M,S)
project3_2,switch3_2 = cal_Q3(N,M,S,P,T,N_p)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project3_2.csv',project3_2,delimiter=',',fmt='%d')
np.savetxt('switch3_2.csv',switch3_2,delimiter=',',fmt='%d')
# 附件 3 数据
print('案例 3 求解')
N = 20  # 包装种类数量
M = 50  # 墨盒数量
S = 10  # 插槽数
P,T,N_p = read_data('Ins3_20_50_10.xlsx',N,M,S
project3_3,switch3_3 = cal_Q3(N,M,S,P,T,N_p)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project3_3.csv',project3_3,delimiter=',',fmt='%d')
np.savetxt('switch3_3.csv',switch3_3,delimiter=',',fmt='%d')
# 附件 4 数据
```

```
print('案例 4 求解')
N = 30  # 包装种类数量
M = 60  # 墨盒数量
S = 10  # 插槽数量
P,T,N_p = read_data('Ins4_30_60_10.xlsx',N,M,S)
project3_4,switch3_4 = cal_Q3(N,M,S,P,T,N_p)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project3_4.csv',project3_4,delimiter=',',fmt='%d')
np.savetxt('switch3_4.csv',switch3_4,delimiter=',',fmt='%d')
```

代码4

介绍：用Python设计Gurobi方案对第4题求解

```
import pulp
import numpy as np
import pandas as pd
import re
import openpyxl
def cal_Q4(N,M,S,P,T,N_p):
    # 创建问题
    prob = pulp.LpProblem("Minimize_Switching_Time", pulp.LpMinimize)
    # 决策变量：x_qijk
    x = pulp.LpVariable.dicts("x", ((q, j, k) for q in range(N) for j in
range(S) for k in range(1, M+1)), 0, 1, pulp.LpBinary)
    # 辅助变量：y_qijkk'
    y = pulp.LpVariable.dicts("y", ((q, j, k_prime, k) for q in range(N) for j
in range(S) for k_prime in range(1, M+1) for k in range(1, M+1)), 0, 1,
pulp.LpBinary)
    # 辅助变量：z_qi 表示第 q 次打印 i 包装
    z = pulp.LpVariable.dicts("z", ((q, i) for q in range(N) for i in
range(N)), 0, 1, pulp.LpBinary)
    # 辅助变量
    # 目标函数：最小化总切换时间
    prob += pulp.lpSum(T[k_prime-1][k-1] * y[q, j, k_prime, k] for q in
range(N) for j in range(S) for k_prime in range(1, M+1) for k in range(1,
M+1))
    # 约束条件
    for q in range(N):
            for j in range(S):
                # 每次印刷每个插槽只能放一个墨盒
                prob += pulp.lpSum(x[q, j, k] for k in range(1, M+1)) <= 1
    for i in range(N):
        for q in range(N):
        # 每个包装的墨盒集合必须满足其需求
            for k in P[i]:
                prob += pulp.lpSum(x[q, j, k] for j in range(S)) >= z[q,i]
    # 插槽沾染颜色后便一直会沾染一种颜色
    for q in range(1, N):
        for j in range(S):
```

```
            prob += pulp.lpSum(x[q-1,j,k] for k in range(1,M+1)) <=
pulp.lpSum(x[q,j,k] for k in range(1,M+1))
    # 定义切换操作
    for q in range(1, N):
            for j in range(S):
                for k_prime in range(1, M+1):
                    for k in range(1, M+1):
                        prob += y[q, j, k_prime, k] >= x[q-1 , j, k_prime] +
x[q, j, k] - 1
    # 满足墨盒顺序
    inf = 999999999
    for i in range(N):
        for q in range(N):
            for j1 in range(S):
                for l in range(1,N_p[i]):
                    k1 = P[i][l-1]
                    k2 = P[i][l]
                    prob += pulp.lpSum(x[q,j,k2] for j in range(j1)) -
pulp.lpSum(x[q,j,k1] for j in range(j1)) <= (1-z[q,i]) * inf
    # 定义顺序
    for q in range(N):
        prob += pulp.lpSum(z[q,i] for i in range(N)) == 1
    for i in range(1,N):
        prob += pulp.lpSum(z[q,i] for q in range(N)) == 1
    # 求解问题
    solver = pulp.GUROBI_CMD(timeLimit=7200,threads=12,msg=0)
    prob.solve(solver)
    # 输出结果
    print("Status:", pulp.LpStatus[prob.status])
    print("最少切换时间:", pulp.value(prob.objective))
    sequence = np.zeros((1,N))
    project = np.zeros((N,S+1))
    switch = []
   #  z
    for q in range(N):
        for i in range(N):
            if pulp.value(z[q,i]) == 1:
                sequence[0,q] = int(i+1)
                # print(f"第{q+1}次打印 -> 包装{i+1}")
    print('--------------------------------------------------')
    # x
    for q in range(N):
        for j in range(S):
            for k in range(1, M+1):
                if pulp.value(x[q, j, k]) == 1:
                    project[q,0] = sequence[0][q]
                    project[q,j+1] = k
```

```
                    # print(f"第 {q+1}次：包装{sequence[0][q]} 插槽 {j+1} -> 墨盒
{k}")
                    break

    print('-------------------------------------------------')
    # 输出 y
    for q in range(1,N):
        for j in range(S):
            for k_prime in range(1,M+1):
                for k in range(1,M+1):
                    if pulp.value(y[q, j,k_prime,k]) == 1:
                        if k_prime != k:
                            switch.append([q+1,sequence[0][q],j+1,k_prime,k])
                            # print(f"第 {q+1}次：包装{sequence[0][q]} 插槽 {j+1}
->从墨盒{k_prime} 更换为{k}")
    switch = np.array(switch)
    return project,switch
def read_data(file_path,N,M,S):
    # path = 'D:\\git 仓库\\machinelearning_ZZLclass\\0531\\附件数据（B 题）\\附
件 4\\' + file_path
    path = 'D:\\Git 仓库\\machinelearning_ZZLclass\\0531\\附件数据（B 题）\\附件
4\\' + file_path
    data_1 = pd.read_excel(path, sheet_name='包装种类及其所需墨盒')
    data_2 = pd.read_excel(path, sheet_name='墨盒切换时间')
   print_info = []
    for i in range(N):
        num = re.findall(r'\d+', data_1.iloc[i, 1])
        print_info.append([int(j) for j in num])
    data_1 = print_info
    N_p = [len(data_1[i]) for i in range(N)]
    data_2 = data_2.values[:,1:]
    return data_1,data_2,N_p
 案例 1 数据
print('案例 1 求解')
N = 5  # 包装种类数量
M = 10  # 墨盒数量
S = 2  # 插槽数量
# 获取 PT 数据
P,T,N_p = read_data('Ins1_5_10_2.xlsx',N,M,S)
project4_1,switch4_1 = cal_Q4(N,M,S,P,T,N_p)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project4_1.csv',project4_1,delimiter=',',fmt='%d')
np.savetxt('switch4_1.csv',switch4_1,delimiter=',',fmt='%d')
# 案例 2 数据
print('案例 2 求解')
N = 5  # 包装种类数量
M = 10  # 墨盒数量
S = 3  # 插槽数量
```

```
# 获取 PT 数据
P,T,N_p = read_data('Ins2_5_10_3.xlsx',N,M,S)
project4_2,switch4_2 = cal_Q4(N,M,S,P,T,N_p)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project4_2.csv',project4_2,delimiter=',',fmt='%d')
np.savetxt('switch4_2.csv',switch4_2,delimiter=',',fmt='%d')
# 案例 3 数据
print('案例 3 求解')
N = 7  # 包装种类数量
M = 10  # 墨盒数量
S = 2  # 插槽数量
# 获取 PT 数据
P,T,N_p = read_data('Ins3_7_10_2.xlsx',N,M,S)
project4_3,switch4_3 = cal_Q4(N,M,S,P,T,N_p)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project4_3.csv',project4_3,delimiter=',',fmt='%d')
np.savetxt('switch4_3.csv',switch4_3,delimiter=',',fmt='%d')
# 案例 4 数据
print('案例 4 求解')
N = 20  # 包装种类数量
M = 40  # 墨盒数量
S = 10  # 插槽数量
# 获取 PT 数据
P,T,N_p = read_data('Ins4_20_40_10.xlsx',N,M,S)
project4_4,switch4_4 = cal_Q4(N,M,S,P,T,N_p)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project4_4.csv',project4_4,delimiter=',',fmt='%d')
np.savetxt('switch4_4.csv',switch4_4,delimiter=',',fmt='%d')
```

代码5

介绍：用Python设计Gurobi方案对第5题求解

```
import pulp
import numpy as np
import pandas as pd
import re
import openpyxl
def cal_Q5(N,M,S,P,T,N_p,n_c):
    # 创建问题
    prob = pulp.LpProblem("Minimize_Switching_Time", pulp.LpMinimize)
    # 决策变量：x_qijk
    x = pulp.LpVariable.dicts("x", ((q, j, k) for q in range(N) for j in
range(S) for k in range(1, M+1)), 0, 1, pulp.LpBinary)
    # 辅助变量：y_qijkk' 表示在第 i 个包装的第 j 个插槽中是否从墨盒 k 切换到墨盒 k'
    y = pulp.LpVariable.dicts("y", ((q, j, k_prime, k) for q in range(N) for j
in range(S) for k_prime in range(1, M+1) for k in range(1, M+1)), 0, 1,
pulp.LpBinary)
    # 辅助变量：z_qi 表示第 q 次打印 i 包装
    z = pulp.LpVariable.dicts("z", ((q, i) for q in range(N) for i in
range(N)), 0, 1, pulp.LpBinary)
```

```
    # 辅助变量 w
    w = pulp.LpVariable.dicts("w", ((q, c, k_prime, k) for q in range(N) for c
in range(n_c) for k_prime in range(1, M+1) for k in range(1, M+1)), 0, 1,
pulp.LpBinary)
    # 辅助变量 u
    u = pulp.LpVariable.dicts("u", ((q) for q in range(N)))
    # 辅助变量
    # 目标函数：最小化总切换时间
    prob += pulp.lpSum(u[q] for q in range(N))
    # 约束条件
    for q in range(N):
            for j in range(S):
                # 每次印刷每个插槽只能放一个墨盒
                prob += pulp.lpSum(x[q, j, k] for k in range(1, M+1)) <= 1
    for i in range(N):
        for q in range(N):
        # 每个包装的墨盒集合必须满足其需求
            for k in P[i]:
                prob += pulp.lpSum(x[q, j, k] for j in range(S)) >= z[q,i]
    # 插槽沾染颜色后便一直会沾染一种颜色
    for q in range(1, N):
        for j in range(S):
            prob += pulp.lpSum(x[q-1,j,k] for k in range(1,M+1)) <=
pulp.lpSum(x[q,j,k] for k in range(1,M+1))
    # 定义切换操作
    for q in range(1, N):
            for j in range(S):
                for k_prime in range(1, M+1):
                    for k in range(1, M+1):
                        prob += y[q, j, k_prime, k] >= x[q-1 , j, k_prime] +
x[q, j, k] - 1
    # 满足墨盒顺序
    inf = 999999999
    for i in range(N):
        for q in range(N):
            for j1 in range(S):
                for l in range(1,N_p[i]):
                    k1 = P[i][l-1]
                    k2 = P[i][l]
                    prob += pulp.lpSum(x[q,j,k2] for j in range(j1)) -
pulp.lpSum(x[q,j,k1] for j in range(j1)) <= (1-z[q,i]) * inf
    # 定义顺序
    for q in range(N):
        prob += pulp.lpSum(z[q,i] for i in range(N)) ==
    for i in range(1,N):
        prob += pulp.lpSum(z[q,i] for q in range(N)) == 1
    # 清洗机分配
    for q in range(N):
```

```
        for j in range(S):
            for k_prime in range(1,M+1):
                for k in range(1,M+1):
                    prob += pulp.lpSum(w[q,c,k_prime,k] for c in range(n_c)) >=
y[q,j,k_prime,k]
    # 每个清洗任务分配一个清洗机
    for q in range(N):
        for k_prime in range(1,M+1):
            for k in range(1,M+1):
                prob += pulp.lpSum(w[q,c,k_prime,k] for c in range(n_c)) <= 1
    # 每次清洗时间
    for q in range(N):
        for c in range(n_c):
            prob += pulp.lpSum(w[q,c,k_prime,k] * T[k_prime-1][k-1] for k_prime
in range(1, M+1) for k in range(1, M+1)) <= u[q]
    # 求解问题
    solver = pulp.GUROBI_CMD(timeLimit=7200,threads=12,msg=0)
    prob.solve(solver)
    # 输出结果
    print("Status:", pulp.LpStatus[prob.status])
    print("最少切换时间:", pulp.value(prob.objective))
    sequence = []
    project = np.zeros((N,S+1))
    switch = []
    #  z
    for q in range(N):
        for i in range(N):
            if pulp.value(z[q,i]) == 1:
                sequence.append(int(i+1))
                # print(f"第{q+1}次打印 -> 包装{i+1}")
    print('-------------------------------------------------')
    # x
    for q in range(N):
        for j in range(S):
            for k in range(1, M+1):
                if pulp.value(x[q, j, k]) == 1:
                    project[q,0] = sequence[q]
                    project[q,j+1] = k
                    # print(f"第 {q+1}次：包装{sequence[q]} 插槽 {j+1} -> 墨盒
{k}")
                    break
    print('-------------------------------------------------')
    # 输出 y
    for q in range(N):
        for j in range(S):
            for k_prime in range(1,M+1):
                for k in range(1,M+1):
                    if k_prime != k and pulp.value(y[q,j,k_prime,k]) == 1:
```

```
                        for c in range(n_c):
                            if pulp.value(w[q,c,k_prime,k]) == 1:

switch.append([q+1,sequence[q],c+1,j+1,k_prime,k])
                                # print(f"第 {q+1}次：包装{sequence[q]} 插槽
{j+1},清洗机{c+1} ->从墨盒{k_prime} 更换为{k}")
                                break
                        break
    switch = np.array(switch)
    sequence = np.array(sequence)
    return project,switch

def read_data(file_path,N,M,S):
    path = 'D:\\git 仓库\\machinelearning_ZZLclass\\0531\\附件数据（B 题）\\附件
5\\' + file_path
    # path = 'D:\\Git 仓库\\machinelearning_ZZLclass\\0531\\附件数据（B 题）\\附
件 3\\' + file_path
    data_1 = pd.read_excel(path, sheet_name='包装种类及其所需墨盒')
    data_2 = pd.read_excel(path, sheet_name='墨盒切换时间'
    print_info = []
    for i in range(N):
        num = re.findall(r'\d+', data_1.iloc[i, 1])
        print_info.append([int(j) for j in num])
    data_1 = print_info
    N_p = [len(data_1[i]) for i in range(N)]
    data_2 = data_2.values[:,1:]
    return data_1,data_2,N_p
# 案例 1 数据
print('案例 1 求解')
N = 5  # 包装种类数量
M = 5  # 墨盒数量
S = 2  # 插槽数量
# 获取 PT 数据
P,T,N_p = read_data('Ins1_5_5_2.xlsx',N,M,S)
project5_1,switch5_1 = cal_Q5(N,M,S,P,T,N_p,2)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project5_1.csv',project5_1,delimiter=',',fmt='%d')
np.savetxt('switch5_1.csv',switch5_1,delimiter=',',fmt='%d')
# 案例 2 数据
print('案例 2 求解')
N = 10  # 包装种类数量
M = 50  # 墨盒数量
S = 15  # 插槽数量
# 获取 PT 数据
P,T,N_p = read_data('Ins2_10_50_15.xlsx',N,M,S)
project5_2,switch5_2 = cal_Q5(N,M,S,P,T,N_p,2)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project5_2.csv',project5_2,delimiter=',',fmt='%d')
```

```
np.savetxt('switch5_2.csv',switch5_2,delimiter=',',fmt='%d')
# 案例 3 数据
print('案例 3 求解')
N = 20  # 包装种类数量
M = 50  # 墨盒数量
S = 10  # 插槽数量
# 获取 PT 数据
P,T,N_p = read_data('Ins3_20_50_10.xlsx',N,M,S)
project5_3,switch5_3 = cal_Q5(N,M,S,P,T,N_p,2)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project5_3.csv',project5_3,delimiter=',',fmt='%d')
np.savetxt('switch5_3.csv',switch5_3,delimiter=',',fmt='%d')
# 案例 4 数据
print('案例 4 求解')
N = 30  # 包装种类数量
M = 60  # 墨盒数量
S = 10  # 插槽数量
# 获取 PT 数据
P,T,N_p = read_data('Ins4_30_60_10.xlsx',N,M,S)
project5_4,switch5_4 = cal_Q5(N,M,S,P,T,N_p,2)
# 把 project 和 switchcar 存为 csv 文件
np.savetxt('project5_4.csv',project5_4,delimiter=',',fmt='%d')
np.savetxt('switch5_4.csv',switch5_4,delimiter=',',fmt='%d')
```

代码 6

介绍：用 Python 设计遗传算法方案对第 2 题求解

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

class Problem:
    def __init__(self, box_num, package_num, slot_num, print_info, clean_time, population_size, generations, cross_rate,
                 mutation_rate):
        self.b_num = box_num
        self.m = package_num
        self.n = slot_num
        self.print_info = print_info
        self.T = clean_time
        self.population_size = population_size
        self.maxgen = generations
        self.cross_rate = cross_rate
        self.mutation_rate = mutation_rate

    def initialize_population(self):
        """
        初始化种群
        individual:插槽被墨盒染色的编号组成的二维矩阵
```

```
                """
                population = []
                for _ in range(self.population_size):
                    indi = [0] * self.n
                    individual = np.zeros((self.m, self.n))
                    for i in range(package_num):
                        index = random.sample(range(self.n), len(self.print_info[i]))
                        for j in range(len(index)):
                            indi[index[j]] = print_info[i][j]
                        individual[i, :] = indi
                    population.append(individual)
                return population

            def calculate_fitness(self, individual):
                """计算适应度值，问题中为插槽清洗次数越小越好"""
                clean_time = 0
                individual = individual.astype(int)
                for i in range(self.m - 1):
                    index = individual[i + 1, :] != individual[i, :]
                    clean_time += np.sum(self.T[individual[i, index], individual[i + 1,
index]])
                fitness = 1 / clean_time
                return fitness

            def selection(self, population):
                """选择操作，二元锦标赛"""
                selected = []
                for _ in range(self.population_size):
                    individual1 = random.choice(population)
                    individual2 = random.choice(population)
                    if self.calculate_fitness(individual1) >
self.calculate_fitness(individual2):
                        selected.append(individual1)
                    else:
                        selected.append(individual2)
                return selected

            def cross_check(self, individual, cross_column):
                for i in range(self.m):
                    for element in self.print_info[i]:
                        if np.isin(element, individual[i]):
                            pass
                        else:
                            individual[i, cross_column] = element
                # 找到数组中的重复元素及其索引
                unique_elements, inverse_indices, counts = np.unique(individual[0, :],
return_inverse=True, return_counts=True)
                duplicate_indices = np.where(counts > 1)[0]
                # 对于每个重复元素，将其索引的其中一个更改为 0
```

```
                for idx in duplicate_indices:
                    duplicate_values = unique_elements[idx]
                    duplicate_idxs = np.where(individual[0, :] == duplicate_values)[0]
                    # 将重复元素的其中一个索引更改为 0
                    individual[0, duplicate_idxs[1]] = 0
                return individual

            def crossover(self, parent1, parent2):
                """交叉"""
                child1 = parent1.copy()
                child2 = parent2.copy()
                if random.random() < self.cross_rate:
                    crossover_column = random.randint(0, self.n - 1)
                    child1[:, crossover_column] = parent2[:, crossover_column]
                    child2[:, crossover_column] = parent1[:, crossover_column]
                    self.cross_check(child1, crossover_column)
                    self.cross_check(child2, crossover_column)
                return child1, child2

            def mutate(self, individual):
                """变异"""
                if random.random() < self.mutation_rate:
                    i = random.randint(0, self.m - 1)
                    idx1, idx2 = random.sample(range(self.n), 2)
                    point1 = individual[i, idx1].copy()
                    point2 = individual[i, idx2].copy()
                    individual[i, idx1], individual[i, idx2] = point2, point1
                return individual

            def genetic_algorithm(self):
                """主函数"""
                best_fitness = []
                best_individual = []
                t1 = int(round(time.time() * 1000))
                population = self.initialize_population()
                for generation in range(self.maxgen):
                    population = sorted(population, reverse=True, key=lambda x:
self.calculate_fitness(x))
                    selected = self.selection(population)
                    next_population = []
                    while len(next_population) < self.population_size:
                        parent1, parent2 = random.sample(selected, 2)
                        child1, child2 = self.crossover(parent1, parent2)
                        next_population.append(self.mutate(child1))
                        next_population.append(self.mutate(child2))
                    best_individual.append(max(population, key=lambda x:
self.calculate_fitness(x)))
                    best_fitness.append(self.calculate_fitness(best_individual[-1]))
                    population = next_population
```

```
            t2 = int(round(time.time() * 1000))
            Solution_time = t2 - t1
            return best_individual, best_fitness, Solution_time


    def keep_first_occurrence(arr):
        # 遍历数组，只保留第一次出现的重复元素
        for i in range(arr.shape[0]):
            seen = set()
            for j in range(arr.shape[1]):
                if arr[i, j] in seen:
                    arr[i, j] = 0
                else:
                    seen.add(arr[i, j])
        return arr


    data_1 = pd.read_excel(
        r'C:\Users\25492\Desktop\2024杭电第14届研究生数学建模竞赛赛题\2024杭电数模校赛——研究生14届赛题B题\附件数据（B题）\附件2\Ins2_7_10_2.xlsx',
        sheet_name='包装种类及其所需墨盒')
    data_2 = pd.read_excel(
        r'C:\Users\25492\Desktop\2024杭电第14届研究生数学建模竞赛赛题\2024杭电数模校赛——研究生14届赛题B题\附件数据（B题）\附件2\Ins2_7_10_2.xlsx',
        sheet_name='墨盒切换时间', index_col=0)
    CleanTime = np.array(data_2)
    CleanTime = np.vstack((np.array([0] * CleanTime.shape[1]), CleanTime))
    CleanTime = np.hstack((np.array([0] * CleanTime.shape[0]).reshape(-1, 1), CleanTime))
    package_num = 7
    box_num = 10
    slot_num = 2
    print_info = []
    for i in range(package_num):
        num = re.findall(r'\d+', data_1.iloc[i, 1])
        print_info.append([int(j) for j in num])
    # 遗传算法参数
    population_size = 100
    generations = 1000
    mutation_rate = 0.1
    cross_rate = 0.8
    problem = Problem(box_num, package_num, slot_num, print_info, CleanTime, population_size, generations, cross_rate,
                      mutation_rate)
    best_individual, best_fitness, Solution_time = problem.genetic_algorithm()
    solution = best_individual[np.argmax(best_fitness)]
    box_solution = keep_first_occurrence(solution)
    result = np.around(1 / np.max(best_fitness))
    print('Ins2_7_10_2')
```

```
    print('墨盒放置方案：\n', box_solution)
    print('最小清洗时间：', result)
    print('计算时间：', Solution_time)
    df = pd.DataFrame(box_solution)
df.to_csv('问题二 Ins2_7_10_2 遗传算法解.csv')
```

代码 7

介绍：用 Python 设计遗传算法方案对第 5 题求解

```
    import random
    import pandas as pd
    import re
    import numpy as np
    import time
    import greedy


    class Problem1:
        def __init__(self, box_num, package_num, slot_num, print_info, clean_time, population_size, generations, cross_rate,
                     mutation_rate):
            self.b_num = box_num
            self.m = package_num
            self.n = slot_num
            self.print_info = print_info
            self.T = clean_time
            self.population_size = population_size
            self.maxgen = generations
            self.cross_rate = cross_rate
            self.mutation_rate = mutation_rate

        def initialize_population(self):
            """
            初始化种群
            individual:包含编码和包装顺序的列表
            """
            population = []
            for _ in range(self.population_size):
                indi = [0] * self.n
                individual = []
                pack_list = list(range(self.m))
                random.shuffle(pack_list)
                for i in pack_list:
                    index = sorted(random.sample(range(self.n), len(self.print_info[i])))
                    for j in range(len(index)):
                        indi[index[j]] = print_info[i][j]
                    individual.append(indi.copy())  # 下一次打印插槽若没有插入其他墨盒则继承上一个墨盒的颜色
                population.append([np.array(individual), pack_list])  # 后面的individual变成了包含染色体编码和包装顺序的列表
```

```
                return population

        def calculate_fitness(self, individual):
            """计算适应度值，问题中为插槽清洗次数越小越好"""
            clean_time = 0
            individual = individual[0].astype(int)
            for i in range(self.m - 1):
                index = individual[i + 1, :] != individual[i, :]
                shift_time = self.T[individual[i, index], individual[i + 1, index]]
                _, _, completion_time = greedy.solve(shift_time)
                clean_time += completion_time
            fitness = 1 / clean_time
            return fitness
        def selection(self, population):
            """选择操作，二元锦标赛"""
            selected = []
            for _ in range(self.population_size):
                individual1 = random.choice(population)
                individual2 = random.choice(population)
                if self.calculate_fitness(individual1) > self.calculate_fitness(individual2):
                    selected.append(individual1)
                else:
                    selected.append(individual2)
            return selected
        def sequence_check(self, individual):
            for i in range(self.m):
                index = []
                for element in print_info[individual[1][i]]:
                    index.append(np.where(individual[0][i, :] == element)[0][0])
                for j in range(len(index) - 1):
                    if index[j] > index[j + 1]:
                        individual[0][i, j], individual[0][i, j + 1] = individual[0][i, j + 1], individual[0][i, j]
                        index[j], index[j + 1] = index[j + 1], index[j]
            return individual
        def initial_check(self, individual):
            """第一组打印墨盒检测，此时应该不存在重复的颜色"""
            # 检测第一行重复元素
            unique_elements, inverse_indices, counts = np.unique(individual[0][0, :], return_inverse=True,

return_counts=True)
            duplicate_indices = np.where(counts > 1)[0]
            # 对于每个重复元素，将其索引的其中一个更改为 0
            for idx in duplicate_indices:
                duplicate_values = unique_elements[idx]
                duplicate_idxs = np.where(individual[0][0, :] == duplicate_values)[0]
                # 将重复元素的其中一个索引更改为 0
```

```
                    individual[0][0, duplicate_idxs[1]] = 0
            return individual
        def cross_check(self, individual, cross_column):
            for i in range(self.m):
                for element in self.print_info[individual[1][i]]:
                    if np.isin(element, individual[0][i]):
                        pass
                    else:
                        individual[0][i, cross_column] = element
            self.initial_check(individual)
            self.sequence_check(individual)
            return individual
        def crossover(self, parent1, parent2):
            """交叉"""
            child1 = parent1.copy()
            child2 = parent2.copy()
            if random.random() < self.cross_rate:
                crossover_column = random.randint(0, self.n - 1)
                child1[0][:, crossover_column] = parent2[0][:, crossover_column]   #
列交叉
                child2[0][:, crossover_column] = parent1[0][:, crossover_column]
                self.cross_check(child1, crossover_column)
                self.cross_check(child2, crossover_column)
            return child1, child2
        def mutation_check(self, individual):
            """检查变异后是否存在颜色不继承"""
            for j in range(self.n):
                for i in range(self.m - 1):
                    if individual[0][i + 1, j] == 0 and individual[0][i, j] != 0:
                        individual[0][i + 1, j] = individual[0][i, j]
            return individual
        def mutate(self, individual):
            """变异"""
            if random.random() < self.mutation_rate:
                idx1, idx2 = random.sample(range(self.m), 2)
                point1 = individual[0][idx1, :].copy()
                point2 = individual[0][idx2, :].copy()
                individual[0][idx1, :], individual[0][idx2, :] = point2, point1   #行交
换
                point1 = individual[1][idx1]
                point2 = individual[1][idx2]
                individual[1][idx1], individual[1][idx2] = point2, point1   # 交换包
装打印顺序
                self.initial_check(individual)
                self.mutation_check(individual)
            return individual
        def genetic_algorithm(self):
            """主函数"""
            best_individual = []
```

```
                best_fitness = []
                t1 = int(round(time.time() * 1000))
                population = self.initialize_population()
                for generation in range(self.maxgen):
                    population = sorted(population, reverse=True, key=lambda x: self.calculate_fitness(x))
                    selected = self.selection(population)
                    next_population = []
                    while len(next_population) < self.population_size:
                        parent1, parent2 = random.sample(selected, 2)
                        child1, child2 = self.crossover(parent1, parent2)
                        next_population.append(self.mutate(child1))
                        next_population.append(self.mutate(child2))
                    best_individual.append(max(population, key=lambda x: self.calculate_fitness(x)))
                    best_fitness.append(self.calculate_fitness(best_individual[-1]))
                    population = next_population
                t2 = int(round(time.time() * 1000))
                Solution_time = t2 - t1
                return best_individual, best_fitness, Solution_time
    def keep_first_occurrence(arr):
        # 遍历数组，只保留第一次出现的重复元素
        for i in range(arr.shape[0]):
            seen = set()
            for j in range(arr.shape[1]):
                if arr[i, j] in seen:
                    arr[i, j] = 0
                else:
                    seen.add(arr[i, j])
        return arr
    data_1 = pd.read_excel(r'C:\ 2024杭电第14届研究生数学建模竞赛赛题\2024杭电数模校赛——研究生14届赛题B题\附件数据（B题）\附件5'
                           r'\Ins3_20_50_10.xlsx', sheet_name='包装种类及其所需墨盒')
    data_2 = pd.read_excel(r'C:\ 2024杭电第14届研究生数学建模竞赛赛题\2024杭电数模校赛——研究生14届赛题B题\附件数据（B题）\附件5'
                           r'\Ins3_20_50_10.xlsx', sheet_name='墨盒切换时间', index_col=0)
    CleanTime = np.array(data_2)
    CleanTime = np.vstack((np.array([0] * CleanTime.shape[1]), CleanTime))
    CleanTime = np.hstack((np.array([0] * CleanTime.shape[0]).reshape(-1, 1), CleanTime))
    """记得改参数!!! """
    package_num = 20
    box_num = 50
    slot_num = 10
    print_info = []
    for i in range(package_num):
        num = re.findall(r'\d+', data_1.iloc[i, 1])
```

```
            print_info.append([int(j) for j in num])
    # 遗传算法参数
    population_size = 100
    generations = 1000
    mutation_rate = 0.3
    cross_rate = 0.8
    problem = Problem1(box_num, package_num, slot_num, print_info, CleanTime,
population_size, generations, cross_rate,
                                mutation_rate)
    best_individual, best_fitness, Solution_time = problem.genetic_algorithm()
    solution = best_individual[np.argmax(best_fitness)]
    box_solution = keep_first_occurrence(solution[0])
    result = np.around(1 / np.max(best_fitness))
    print_sequence = [solution[1][i] + 1 for i in range(package_num)]
    print('Ins3_20_50_10')
    print('墨盒放置方案：\n', box_solution)
    print('最优顺序：', print_sequence)
    print('最小清洗时间：', result)
    print('计算时间：', Solution_time)
    df = pd.DataFrame(box_solution, index=print_sequence)
    df.to_csv('问题五Ins3_20_50_10遗传算法解.csv')
    贪婪算法代码
    def solve(jobs):
        machine1 = []
        machine2 = []
        time_machine1 = 0
        time_machine2 = 0
        for job in sorted(jobs, reverse=True):
            if time_machine1 <= time_machine2:
                machine1.append(job)
                time_machine1 += job
            else:
                machine2.append(job)
                time_machine2 += job
        completion_time = max(time_machine1, time_machine2)
        return machine1, machine2, completion_time
    if __name__ == '__main__':
        # 测试示例
        jobs = [3, 6, 8, 2, 4, 5]
        machine1, machine2, completion_time = solve(jobs)
        print("机器1工件:", machine1)
        print("机器2工件:", machine2)
        print("所有工件完成时间:", completion_time
```

由于本文篇幅有限剩余的代码将在附件中给出