柔性版印刷中墨盒切换方法的优化

摘　要

柔性版印刷技术是一种在印刷行业中被广泛采用的印刷方式，是一种公认的绿色印刷技术。然而由于其使用印刷辊滚动携带染料的特点，使得在墨盒插槽有限，或印刷需求较为复杂的情况下需要频繁更换墨盒并清洗印刷滚筒，这成为限制该技术印刷效率的主要原因之一。本文通过建立优化模型，基于不同情形下的印刷需求对柔性版印刷机的墨盒更换方案进行优化。

对于问题一，我们建立了一个**非线性 0-1 整数规划模型**，目标函数是最小化墨盒切换次数 N_{total}，并使用 Gurobi 求解器进行求解得到附件一中每个实例的优化结果。附件 1 中各样例 N_{total} 的最优值计算结果如下：$N_1 = 5,\ N_2 = 8,\ N_3 = 48,\ N_4 = 58,\ N_5 = 130$。

对于问题二，为求出最小切换时间 T_{total}，我们在原有模型的基础上，构建出不同墨盒间切换所需的时间矩阵，将其替换原有的次数矩阵作为目标函数。为提高求解效率，我们使用**惩罚数法（大 M 法）**对构建的切换时间矩阵进行处理。经过计算，在同一 CPU 运行至 400 秒时，样例 3 的最优可行值已迭代至 112 min，误差率 GAP 降至 14.9%，优于未处理前的 120 min 和 33.7%，表明改进的模型对求解效率有较大提升。由于部分实例的变量规模较大，我们取其较好精度下，并且模型已趋于收敛的可行解作为结果。附件 2 中各样例 T_{total} 的最优值计算结果如下：$T_1 = 32\,\text{min},\ T_2 = 51\,\text{min},\ T_3 = 111\,\text{min},\ T_4 = 210\,\text{min},\ T_5 = 344\,\text{min}$。

对于问题三，我们通过更新模型约束条件，进一步确保每种包装打印时具有固定得墨盒排列次序，满足印刷作业过程中墨盒次序的要求，使得模型可求解在固定墨盒顺序情况下的最小切换时间。同样地，由于部分数据规模较大，我们取其近似最优解作为结果。使用 Gurobi 求解器计算，附件 3 中各样例 T_{total} 的最优值计算结果如下：$T_1 = 56\,\text{min},\ T_2 = 186\,\text{min},\ T_3 = 253\,\text{min},\ T_4 = 266\,\text{min}$。

对于问题四，要求打印包装之前，需要对打印的顺序进行排列，在此条件下再最小化墨盒切换时间，对于其中打印顺序的确定是一个**旅行商问题（TSP）**的变种。通过计算每种包装之间的墨盒切换时间，我们绘制出以包装作为节点的全连通带权有向图，将原规划转化为一个 TSP 问题进行求解。由于 TSP 模型在数据量较大的情况下是一个 NP-Hard 问题，无法直接求解。我们采用**模拟退火算法**求得了一个局部最优解。附件 4 中各样例 T_{total} 的最优值计算结果如下：$T_1 = 33\,\text{min},\ T_2 = 63\,\text{min},\ T_3 = 92\,\text{min},\ T_4 = 767\,\text{min}$

关键词:　最优化问题；0-1 整数规划；旅行商问题；模拟退火算法

目 录

一 前言 **1**

1.1 问题背景 1

1.2 问题重述 1

二 模型准备 **1**

2.1 问题假设 1

2.2 符号说明 2

2.3 主要工作 2

三 问题一：最小化切换次数 **3**

3.1 机理分析 3

3.1.1 确定决策变量 3

3.1.2 刻画约束条件 3

3.1.3 计算目标函数 4

3.2 模型建立 5

3.3 求解结果 6

四 问题二：最小化切换时间 **7**

4.1 构建时间矩阵 7

4.2 模型建立 8

4.3 模型改进 9

4.4 求解结果 10

五 问题三：墨盒相对顺序固定的情形 **11**

5.1 添加顺序约束 11

5.2 模型建立 12

5.3 求解结果 12

六 问题四：包装印刷顺序不定的情形 **13**

6.1 修改目标函数 13

6.2 模型建立 15

6.3 模型求解 15

6.3.1 基于 TSP 问题的求解方法 15

6.3.2 模拟退火算法 19

6.4 求解结果 20

A 附录 A：源代码 **22**

一　前言

1.1　问题背景

现代化的印刷生产中常选用柔性版印刷技术。作为一种公认的绿色印刷方式，柔性版印刷技术具有减少浪费和快速原型设计等多种优势，并在多种不同材质的印刷材料上得到了广泛的使用。柔性版印刷技术的核心是使用反向滚动的印刷辊与印版滚筒将墨盒中的不同颜料压印至材料。同时柔性印刷机的印刷滚轮可以抬起，避免不需要的染料接触印刷材料。由于墨盒在使用时需要事先插入插槽中，且插槽的数量有限，因此在需要更换不同颜色的染料时，需要更换插槽中的染料墨盒并清洗印刷设备，更换过程中所产生的等待时间即是限制印刷效率的主要原因之一。

为了降低更换染料产生的等待时间，我们对不同印染要求下的墨盒且换顺序及印刷材料的印刷次序进行优化，以提高柔性版印刷的整体效率。

1.2　问题重述

某柔性印刷厂的大型印刷机具有 k 个插槽，现需完成一批 j 种打印材料的印刷订单，印刷过程中共使用了 i 个墨盒。现要求对于不同的印刷需求，优化墨盒的更换顺序，在 k, i, j 已知且仅有一套清洗设备的前提下，实现墨盒切换时间的最小化。具体问题如下：

- 固定印刷材料的印刷顺序，且墨盒在插槽中的相对次序对印刷效果无影响，仅考虑墨盒的切换次数，求解出最小化总切换次数的印刷方案。
- 固定印刷材料的印刷顺序，且墨盒在插槽中的相对次序对印刷效果无影响，考虑不同墨盒间的切换时间，求解出最小化总切换时间的印刷方案。
- 固定印刷材料的印刷顺序，但要求每种材料所需的墨盒在插槽中有固定的相对次序，考虑不同墨盒间的切换时间，求解出最小化总切换时间的印刷方案。
- 印刷材料的印刷顺序不固定，且要求每种材料所需的墨盒在插槽中有固定的相对次序，考虑不同墨盒间的切换时间，求解出最小化总切换时间的印刷方案。

二　模型准备

2.1　问题假设

由于生产过程中的实际情况较为复杂，需要结合考虑较多的现实因素，但在模型建立过程中难以完全对这些因素的影响进行统计和量化。为此，我们做出如下假设：

- 保持切换对象的墨盒种类不变（如保持切换 $1 \to 2$ 中的墨盒 1、2 不变），则墨盒间的切换时间不发生改变。
- 每种包装的印刷都在同一批内完成。

- 替换印刷包装种类所需的时间不计算入总时间内。
- 印刷滚轮抬起可看作是一个瞬间的过程，其时间不计算入总时间内。
- 印刷过程中不考虑其它可能事件（如印刷机故障、墨盒中的染料用完等）对总时间产生的影响。

2.2 符号说明

表 1: 符号变量

变量	定义
k	包装类型的编号，$k = 1, 2, \ldots, p$
i	墨盒的编号，$i = 1, 2, \ldots, n$
j	插槽的编号，$j = 1, 2, \ldots, m$
t_{ij}	从墨盒 i 切换到墨盒 j 所需的时间 (单位：min)
N_{total}	总切换次数
T_{total}	总切换时间
C_k	印刷包装 k 所需要的墨盒的集合 $C_k = \{i \mid i$ 为印刷包装 k 所需要的墨盒$\}$

2.3 主要工作

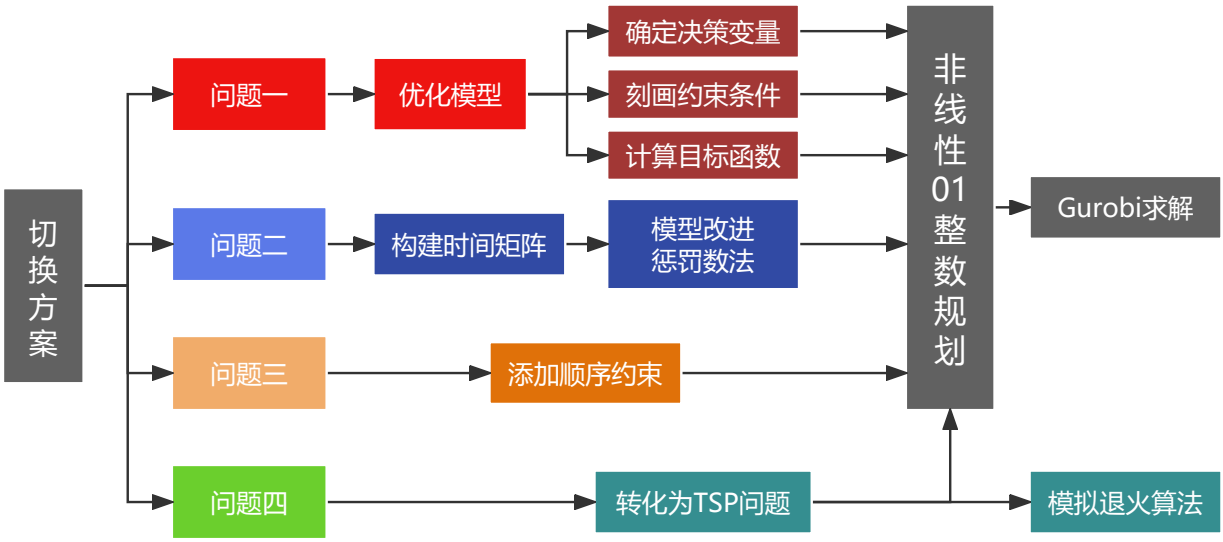


图 1: 模型建立流程

三　问题一：最小化切换次数

3.1　机理分析

3.1.1　确定决策变量

为了直接体现印刷过程中每个插槽中墨盒的切换情况，我们构建了一个三维数组 $X^{p\times n\times m}$，其维度 p, n, m 分别表示**包装类型的总数**、**使用到的墨盒数量**及**插槽个数**。定义决策变量 x_{kij} $(k=1,2,\ldots,p,\ i=1,2,\ldots,n,\ j=1,2,\ldots,m)$ 为

$$x_{kij}=\begin{cases}1, & \text{打印包装 } k \text{ 时，插槽 } j \text{ 内插入墨盒 } i\\ 0, & \text{否则}\end{cases} \tag{1}$$

由此可见 x_{kij} 是 0-1 整型变量，规划求解得到 $p\times n\times m$ 的 0-1 数组，它对于每个不同的包装类型 k 有一个稀疏矩阵 B_k，其中含有 m 个元素为 1，其余元素为 0。

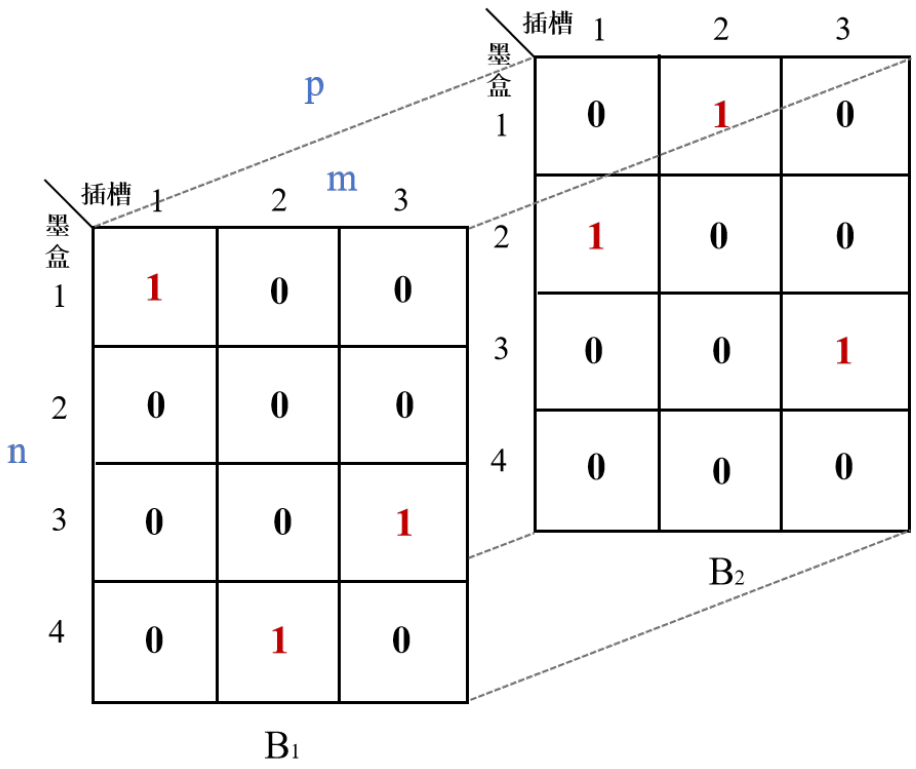


图 2: 决策变量

3.1.2　刻画约束条件

对于我们定义的决策变量 $x_{kij}\in X$，根据印刷过程中的实际需求，可以得到如下约束条件[1]：

(1) 对于特定的包装 k，每个插槽 j 只能插入一个墨盒 i

对特定的包装类型 k，其稀疏矩阵 B_k 共有 m 列，代表柔性版印刷机的 m 个墨盒插槽。因此，若需保证每个插槽只插入一个墨盒，则有

$$\sum_{j=1}^{m} x_{kij} = 1,\ \ k = 1, 2, \ldots, p;\ i = 1, 2, \ldots, n \tag{2}$$

(2) 对于特定的包装 k，每一个墨盒 i 至多只能插在一个插槽 j 内

对特定的包装类型 k，其稀疏矩阵 B_k 共有 n 列，代表柔性版印刷机在此次印刷作业中所使用的全部 n 个墨盒插槽。因此，若需保证每一个墨盒至多只能插在一个插槽内，则有

$$\sum_{i=1}^{n} x_{kij} \leq 1,\ \ k = 1, 2, \ldots, p;\ j = 1, 2, \ldots, m \tag{3}$$

(3) 每种包装 k 都需要有相应颜色的所有墨盒 i

对特定的包装类型 k，我们使用 Python 中的 spilt() 函数从附件数据中获取每种包装对应的所需墨盒 i 的集合 C_k。通过集合 C_k 中的元素定位稀疏 0-1 矩阵 B_k 的每一个所需墨盒对应的行 i。由于印刷机的滚轮可以抬起从而避免印刷到不需要的染料，如矩阵 B_k 中可以出现不属于 C_k 的墨盒对应的非零元素，只需保证所有属于 C_k 的墨盒存在即可。因此，若需保证每种包装 k 都需要有相应颜色的所有墨盒 i，则有

$$\begin{cases} C_k = \{i \mid i \text{ 为印刷包装 } k \text{ 所需要的墨盒}\} \\ \sum\limits_{j=1}^{m} x_{ki'j} = 1,\ \ i' \in C_k;\ k = 1, 2, \ldots, p \end{cases} \tag{4}$$

3.1.3 计算目标函数

为了计算印刷过程中的总切换次数，我们定义了一个 $n \times n$ 维的切换次数矩阵 T，矩阵中的元素 t_{ij} 表示从墨盒 i 切换至墨盒 j 所增加的切换次数。t_{ij} 的定义为

$$t_{ij} = \begin{cases} 1, & i \neq j \\ & , \ i, j = 1, 2, \ldots, n \\ 0, & i = j \end{cases} \tag{5}$$

当 $i \neq j$ 时，插槽中的墨盒从 i 切换至 j，切换次数 $+1$；当 $i = j$ 时，插槽中的墨盒无变化，切换次数保持不变。

我们使用矩阵的乘积来刻画 B_k 至 B_{k+1} 的墨盒变化。对于稀疏矩阵 $B_k,\ B_{k+1}$，我们取

$$S_{k,k+1} = B_k \cdot B_{k+1}^T \tag{6}$$

上述运算式的结果 $S_{k,k+1}$ 为一个 n 阶方阵，其矩阵元素 s_{ij} 为

$$s_{ij} = \begin{cases} 1, & \text{在 } B_k \to B_{k+1} \text{ 中，墨盒 } i \text{ 切换为墨盒 } j \\ 0, & \text{否则} \end{cases}, \quad k = 2, 3, \ldots, p-1 \tag{7}$$

当有墨盒 i 切换至墨盒 j 时，$s_{ij} = 1$；否则，$s_{ij} = 0$。

下图所示为矩阵 $S_{k,k+1}$ 的计算过程：

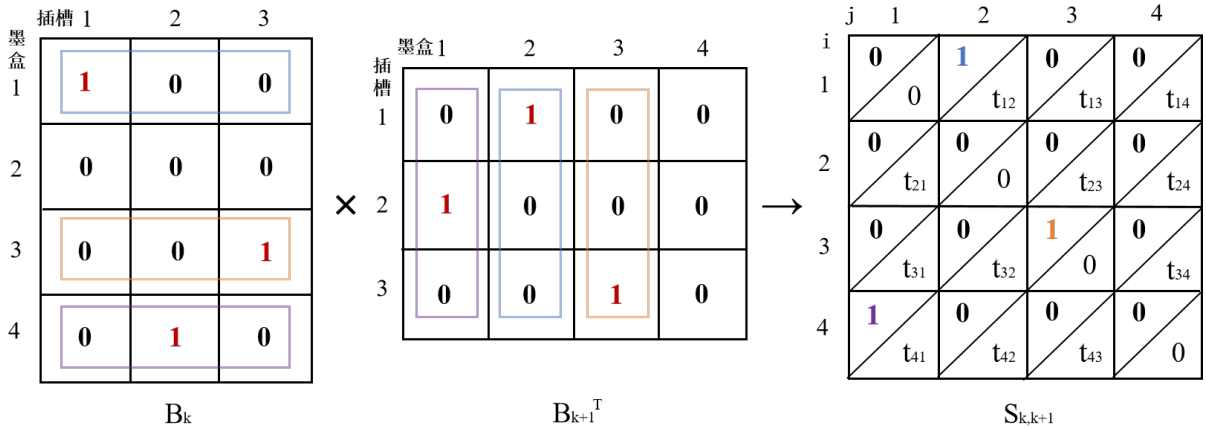


图 3: $S_{k,k+1}$ 的计算过程

我们计算 $S_{k,k+1}$ 与 T 的 Hadamard 积（即矩阵对应元素乘积，记为 $\oplus$）总和，即可得到墨盒切换 $k \to k+1$ 中的次数 N_k。

$$N_k = \sum_{i=1}^{n} \sum_{j=1}^{m} S_{k,k+1} \oplus T \tag{8}$$

则整个印刷过程中的切换总次数 N_{total} 为

$$N_{total} = \sum_{k=1}^{p-1} \sum_{j=1}^{m} \sum_{i(k)=1}^{n} \left[\sum_{i(k+1)=1}^{n} \left(x_{ki^{(k)}j} \cdot x_{(k+1)i^{(k+1)}j} \right) \cdot t_{i^{(k)}i^{(k+1)}} \right] \tag{9}$$

3.2 模型建立

综合上述分析结果，我们可建立求解最小化墨盒切换次数的模型。

$$x_{kij} = \begin{cases} 1, & \text{打印包装 } k \text{ 时，插槽 } j \text{ 内插入墨盒 } i \\ 0, & \text{否则} \end{cases} \tag{10}$$

$$\text{def} \quad C_k = \{i \mid i \text{ 为印刷包装 } k \text{ 所需要的墨盒}\} \tag{11}$$

$$\min \quad T_{total} = \sum_{k=1}^{p-1} \sum_{j=1}^{m} \sum_{i^{(k)}=1}^{n} \left[\sum_{i^{(k+1)}=1}^{n} \left(x_{ki^{(k)}j} \cdot x_{(k+1)i^{(k+1)}j} \right) \cdot t_{i^{(k)}i^{(k+1)}} \right] \tag{12}$$

$$\text{s.t.} \begin{cases} \sum\limits_{j=1}^{m} x_{kij} = 1, \ \ k = 1, 2, \ldots, p; \ i = 1, 2, \ldots, n, \\ \sum\limits_{i=1}^{n} x_{kij} \leq 1, \ \ k = 1, 2, \ldots, p; \ j = 1, 2, \ldots, m \\ \sum\limits_{j=1}^{m} x_{ki'j} = 1, \ \ i' \in C_k; \ k = 1, 2, \ldots, p \\ x_{kij} = 0, \ 1 \end{cases} \tag{13}$$

由于决策变量 x_{kij} 的取值仅为 0 或 1，且目标函数中存在变量矩阵 B_k 与 B_{k+1} 转置的乘积，故该模型为 0-1 非线性整数规划[2]。

3.3 求解结果

从附件 1 中得到待求解模型的样例数据，各样例的模型参数如下表所示：

表 2: 模型一相关数据参数

实例	Ins1	Ins2	Ins3	Ins4	Ins5
包装种类数 p	5	7	10	20	30
墨盒数 n	10	10	50	50	60
插槽数 m	2	3	15	10	10

将相应数据带入模型中，使用 Gurobi 求解器求出最优解。

表 3: 墨盒切换最少次数

实例	Ins1	Ins2	Ins3	Ins4	Ins5
N_{total}	5	8	48	58	130

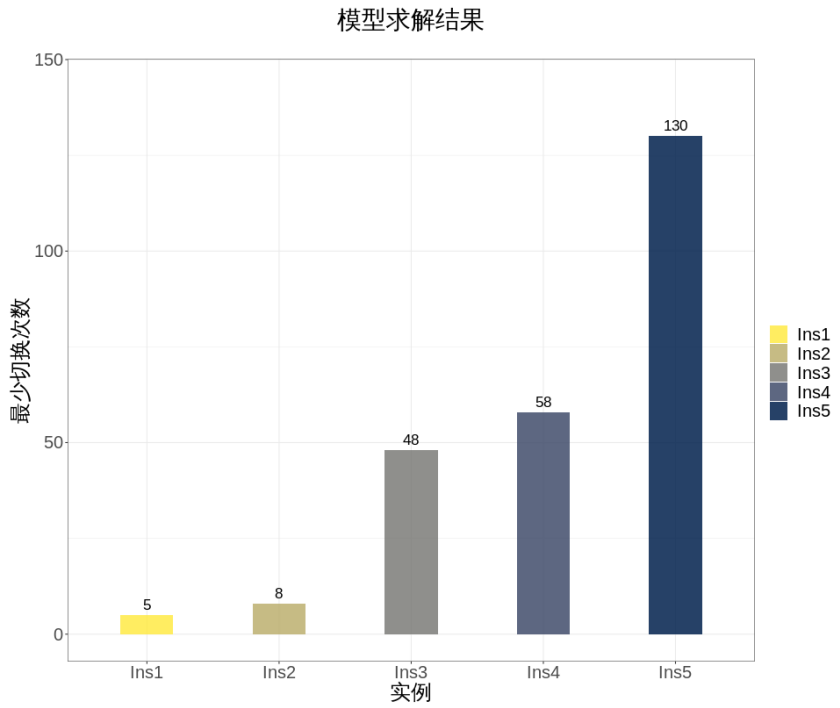


图 4: 模型求解结果

随着包装种类数量，墨盒数量与插槽数量的增加模型的复杂度也成指数形式的增加，需要切换墨盒的次数也在增加，其中墨盒数量的增加对于切换次数增加的影响更大。

四　问题二：最小化切换时间

4.1　构建时间矩阵

基于最小化时间的目的，我们将原模型中的切换次数矩阵 T 进行修改，使其能够反映印刷机在将插槽中的墨盒 i 切换为墨盒 j 所需的时间。矩阵中元素 t'_{ij} 定义为

$$t'_{ij} = \begin{cases} t_{ij}, & i \neq j \\ & , \ i, j = 1, 2, \ldots, n \\ 0, & i = j \end{cases} \tag{14}$$

其中 t_{ij} 为附件样例中墨盒 i 切换至墨盒 j 所需的等待时间。当 $i \neq j$ 时，插槽中的墨盒从 i 切换至 j，需等待的时间 $t'_{ij} = t_{ij}$；当 $i = j$ 时，插槽中的墨盒无变化，$t'_{ij} = 0$，表明无需进行等待。

表 4: 时间矩阵 T' 数据示例

墨盒	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	10	3	15	9	10	10	3	7	8
2	12	0	10	16	4	17	16	15	19	9

续表

墨盒	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
3	12	9	0	15	10	7	8	15	9	9
4	12	9	8	0	11	15	8	12	12	14
5	8	7	11	23	0	18	18	11	15	16
6	5	2	8	9	4	0	1	8	5	2
7	4	1	7	14	3	14	0	7	4	7
8	10	7	13	14	6	18	18	0	4	7
9	10	9	13	10	2	14	14	9	0	3
10	17	14	15	7	13	12	13	17	14	0

4.2 模型建立

将修改后得到的切换时间时间矩阵 T' 代入模型，替换原模型中的切换次数矩阵 T，则可得到求解最小化墨盒切换时间的模型为

$$x_{kij}=\begin{cases}1, & \text{打印包装 } k \text{ 时，插槽 } j \text{ 内插入墨盒 } i\\ 0, & \text{否则}\end{cases} \tag{15}$$

$$\text{def}\quad C_k=\{i \mid i \text{ 为印刷包装 } k \text{ 所需要的墨盒}\} \tag{16}$$

$$\min\quad N_{total}=\sum_{k=1}^{p-1}\sum_{j=1}^{m}\sum_{i^{(k)}=1}^{n}\left[\sum_{i^{(k+1)}=1}^{n}\left(x_{ki^{(k)}j}\cdot x_{(k+1)i^{(k+1)}j}\right)\cdot t'_{i^{(k)}i^{(k+1)}}\right] \tag{17}$$

$$\text{s.t.}\begin{cases}\sum\limits_{j=1}^{m}x_{kij}=1,\ \ k=1,2,\ldots,p;\ i=1,2,\ldots,n,\\ \sum\limits_{i=1}^{n}x_{kij}\le 1,\ \ k=1,2,\ldots,p;\ j=1,2,\ldots,m\\ \sum\limits_{j=1}^{m}x_{ki'j}=1,\ \ i'\in C_k;\ k=1,2,\ldots,p\\ x_{kij}=0,\ 1\end{cases} \tag{18}$$

该模型与问题一中的 0-1 非线性整数规划在形式上没有区别。事实上，问题一中的模型可以看作是修改后模型的特殊情形，即问题一中模型所求的解即为当 T' 中所有 $t'_{ij}=1\ (i\neq j)$ 时的模型的最小值。

4.3 模型改进

在实际求解过程中，由于数据规模过大，求解过程需要耗费大量的时间。为了提高模型的求解效率，我们使用惩罚数法对模型进行改进。

改进的核心思想是在模型求解的过程中尽量避免切换出不需要的墨盒。因此我们选定一个惩罚数 M，保证 M 远大于所有切换时间中的最大值，使得当 $x_{kij}=1$，但 $i \notin C_k$ 时对模型的结果进行惩罚。

在计算墨盒的切换时间矩阵 T' 时，我们令

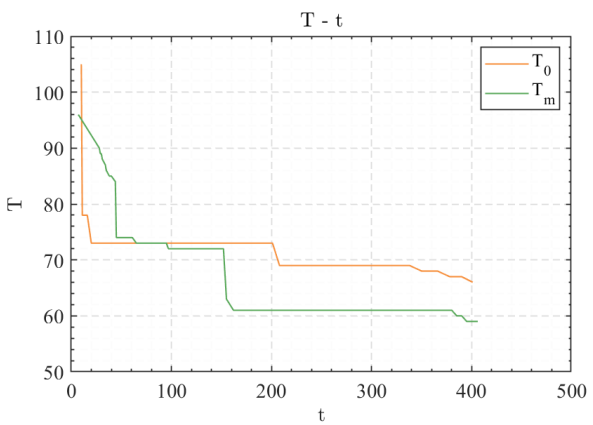
$$t'_{kij}=\begin{cases} t'_{ij}, & i \in C_k \\ M, & i \notin C_k \\ 0, & i=j \end{cases} \tag{19}$$

我们对每类包装 k 计算他们单独的切换时间矩阵 T'_k，其中若墨盒 i 为 k 所需的墨盒，则保持其时间不变，否则，将其切换时间更改为 M。当不属于包装 k 所需墨盒的 i 对应的决策变量 $x_{kij}=1$ 时，总切换时间 T_{total} 将增大 M，从而实现对模型结果的惩罚。

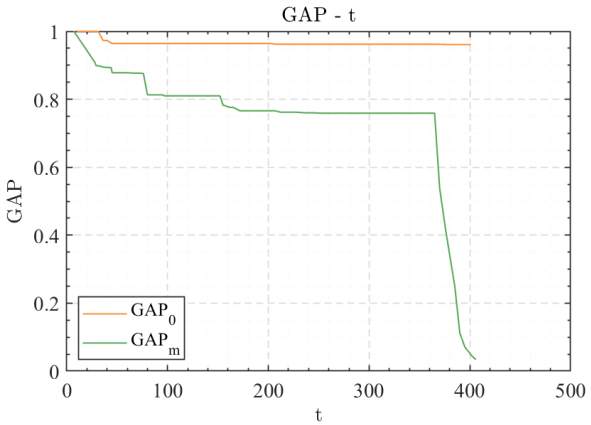
重新计算时间矩阵后，将目标变量改为

$$T_{total}=\sum_{k=1}^{p-1}\sum_{i=1}^{n}\sum_{j=1}^{m} S_{k,k+1} \oplus T'_{k+1} \tag{20}$$

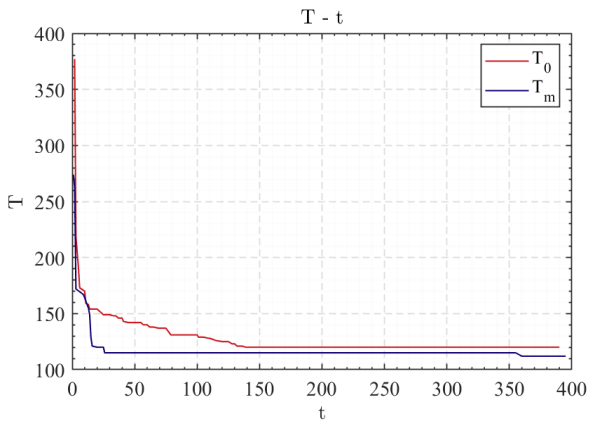
我们绘制在惩罚数引进前后的 Gurobi 求解器在求解该优化模型过程中求得的可行解 T_0, T_m 与误差率 GAP_0, GAP_m 随时间的变化情况，并进行对比。两组数据均放在同一 CPU 上运行，且运行环境基本一致。



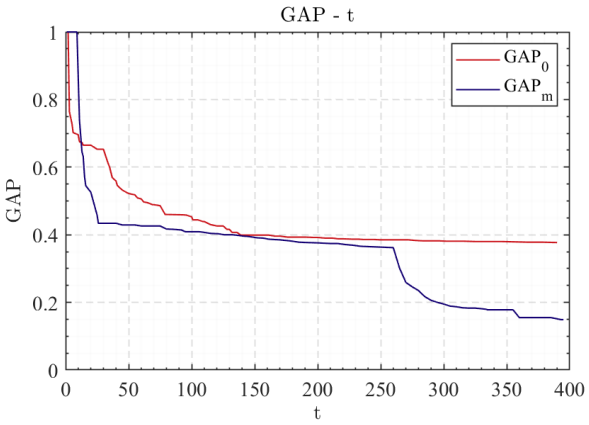
(a) 样例 1.4 T-t



(b) 样例 1.4 GAP-t



(c) 样例 2.3 T-t



(d) 样例 2.3 GAP-t

图 5: 模型改进前后对比

从图像中可以看出，Gurobi 求解器对样例的计算在模型修改后结果 T 和误差率 GAP 的收敛的速度均有了较大的提升，特别是在后半段，可行解与最优解间误差率 GAP 的下降速度明显加快，表明模型改进后的求解效率有了较大的提升。

4.4 求解结果

从附件 2 中得到待求解模型的样例数据，各样例的模型参数如下表所示：

表 5: 模型二相关数据参数

实例	Ins1	Ins2	Ins3	Ins4	Ins5
包装种类数 p	5	7	10	20	30
墨盒数 n	10	10	30	40	60
插槽数 m	3	2	10	10	10

将相应数据带入模型中，使用 Gurobi 求解器求出最优解。

表 6: 墨盒切换最短时间

实例	Ins1	Ins2	Ins3	Ins4	Ins5
T_{total}	32	51	111	210	344

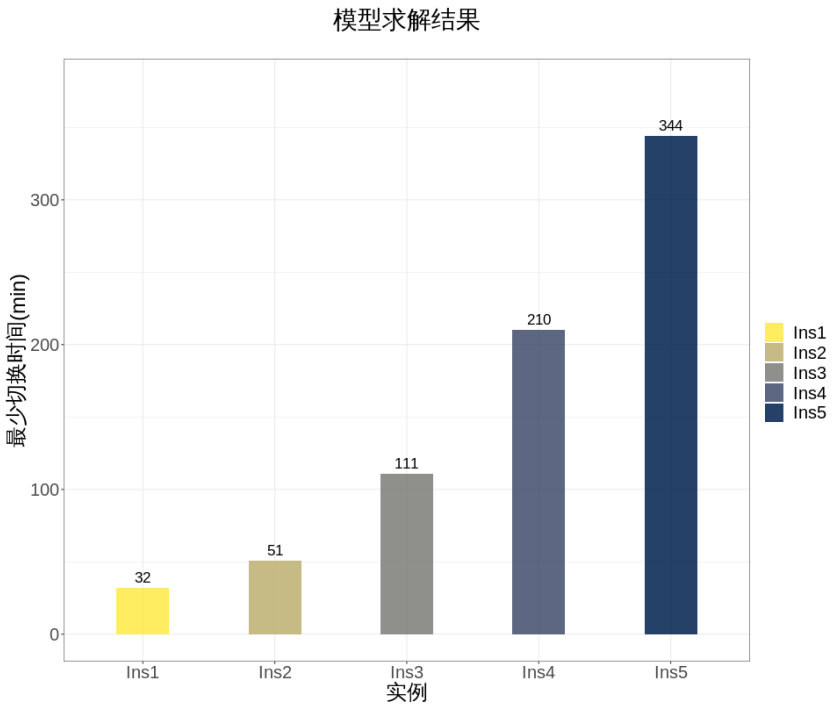


图 6: 模型求解结果

需要说明的是，对于 Ins4 与 Ins5，模型需要很长时间进行求解，但通过观察 Gurobi 的求解过程我们发现，在运行到一定 Gap 值时，可行解不会再发生改变，我们就求这时的解为次优解，最为优化结果。Gap 值是一个衡量求解器性能和接近最优解程度的重要指标，Gap 值越小，表示当前找到的可行解与最优解的差距越小。

五　问题三：墨盒相对顺序固定的情形

5.1　添加顺序约束

由于包装材质和印刷工艺的要求，部分包装对所需的染料的顺序有着相对严格的印刷顺序，因此我们在建立模型求解最小化时间的过程中需要考虑到需要固定墨盒相对顺序的情形。为此我们需要在原有模型的基础上添加约束条件，以保证墨盒顺序的固定。

我们从 C_k 中获取包装类型 k 所需的所有墨盒 i_q，$i_q \in C_k,\ q = 1, 2, \ldots, \tilde{n}$（$\tilde{n}$ 为 C_k 中的元素个数）。随后按要求的先后顺序计算其所在的插槽位置 j_q。

$$j_q = \sum_{j=1}^{m} \left(x_{ki_qj} \cdot j \right),\ q = 1, 2, \ldots, \tilde{n} \tag{21}$$

计算可得到向量 $J_k = (j_1, j_2, \ldots, j_{\tilde{n}})$。其储存了每个包装 k 所需要的墨盒 i_q 对应的插槽位置 j_q。

要保证墨盒的相对次序固定，需要保证向量 J_k 中的元素 j_q 在数值上是递增的。

$$j_q < j_{q+1},\ \ i_q \in C_k;\ j_q \in J_k;\ q = 1, 2, \ldots, \tilde{n} - 1 \tag{22}$$

又可写作

$$\sum_{j=1}^{m}\left(x_{ki_qj}\cdot j\right)<\sum_{j=1}^{m}\left(x_{ki_{q+1}j}\cdot j\right),\ \ q=1,2,\ldots,\tilde{n}-1 \tag{23}$$

5.2 模型建立

添加新的约束后，即可得到在固定墨盒相对顺序情形下求解最小化切换时间的模型为

$$x_{kij}=\begin{cases}1, & \text{打印包装 } k \text{ 时，插槽 } j \text{ 内插入墨盒 } i\\ 0, & \text{否则}\end{cases} \tag{24}$$

$$\text{def}\quad C_k=\{i\mid i \text{ 为印刷包装 } k \text{ 所需要的墨盒}\} \tag{25}$$

$$\text{def}\quad \tilde{n}=\dim\left(C_k\right) \tag{26}$$

$$\min\quad T_{total}=\sum_{k=1}^{p-1}\sum_{j=1}^{m}\sum_{i^{(k)}=1}^{n}\left[\sum_{i^{(k+1)}=1}^{n}\left(x_{ki^{(k)}j}\cdot x_{(k+1)i^{(k+1)}j}\right)\cdot t'_{(k+1)i^{(k)}i^{(k+1)}}\right] \tag{27}$$

$$\text{s.t.}\begin{cases}\sum\limits_{j=1}^{m}x_{kij}=1,\ \ k=1,2,\ldots,p;\ i=1,2,\ldots,n,\\ \sum\limits_{i=1}^{n}x_{kij}\le 1,\ \ k=1,2,\ldots,p;\ j=1,2,\ldots,m\\ \sum\limits_{j=1}^{m}x_{ki'j}=1,\ \ i'\in C_k;\ k=1,2,\ldots,p\\ \sum\limits_{j=1}^{m}\left(x_{ki_qj}\cdot j\right)<\sum\limits_{j=1}^{m}\left(x_{ki_{q+1}j}\cdot j\right),\ \ i_q\in C_k;\ q=1,2,\ldots,\tilde{n}-1\\ x_{kij}=0,\ 1\end{cases} \tag{28}$$

5.3 求解结果

从附件 3 中得到待求解模型的样例数据，各样例的模型参数如下表所示：

表 7: 模型三相关数据参数

实例	Ins1	Ins2	Ins3	Ins4
包装种类数 p	5	10	20	30
墨盒数 n	5	30	50	60
插槽数 m	2	10	10	10

将相应数据带入模型中，使用 Gurobi 求解器求出最优解。

表 8: 墨盒切换最短时间

实例	Ins1	Ins2	Ins3	Ins4
T_{total}	56	186	253	266

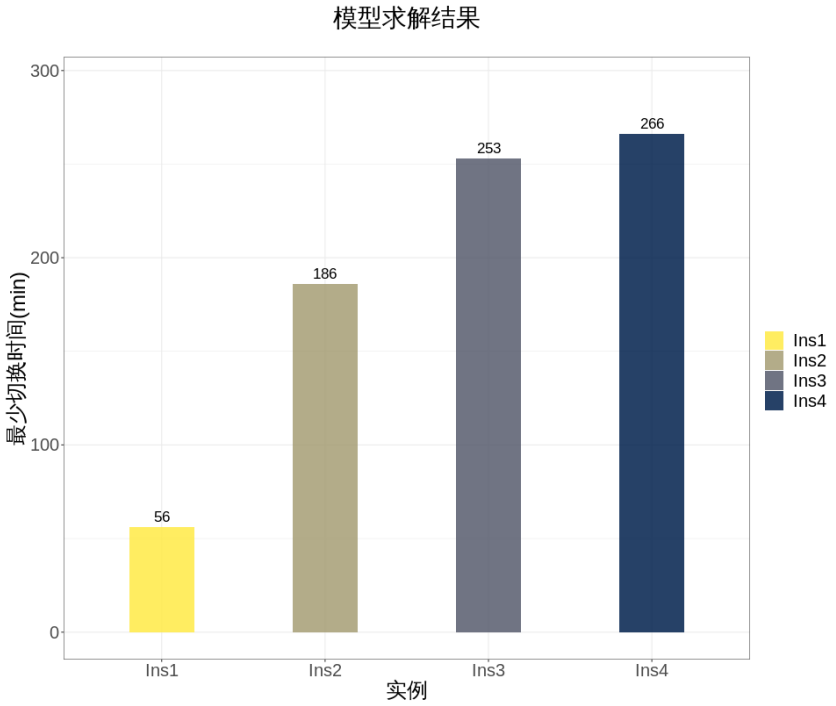


图 7: 模型求解结果

同样的，由于 Ins3 与 Ins4 的求解也需要大量时间，我们取当 Gap 值较小且可行解不再改变的时的解作为次优解，代表模型的结果。

六　问题四：包装印刷顺序不定的情形

6.1　修改目标函数

对于多种不同包装的印刷作业，我们还可以通过调整包装的印刷顺序的方法，进一步优化印刷方案。为此，我们定义了一个 $p \times p$ 维的包装印刷切换矩阵 Y 作为决策变量矩阵。矩阵中元素 y_{ij} 定义为

$$y_{kr} = \begin{cases} 1, & \text{由包装 } k \text{ 切换至包装 } r \\ 0, & \text{否则} \end{cases}, \quad k, r = 1, 2, \ldots, p \tag{29}$$

切换矩阵 Y 中，$y_{ij} = 1$ 表示印刷过程中由矩阵行下标 i 对应的包装切换到列下表 j 对应的包装；而 $y_{ij} = 0$ 则表示不做如此切换。

对于上述切换矩阵 Y，有下列约束：

(1) 对于切换 $B_k \to B_r$，除第一个包装 k_{start} 外每种包装只作为一次切换的起点 k

$$\sum_{r=1}^{p} y_{kr} = 1, \ \ k = 1, 2, \ldots, p, \ k \neq k_{start} \tag{30}$$

(2) 对于切换 $B_k \to B_r$，除最后一个包装 r_{end} 外每种包装只作为一次切换的终点 r

$$\sum_{k=1}^{p} y_{kr} = 1, \ \ r = 1, 2, \ldots, p, \ r \neq r_{end} \tag{31}$$

(3) 包装 B_k 不允许切换为自身

$$y_{kk} = 0, \ \ k = 1, 2, \ldots, p \tag{32}$$

基于问题三中的优化模型，我们能够计算出任意两种包装 k, r 间的理论最小切换时间 $\tilde{t}_{kr\ min}\ (k, r = 1, 2, \ldots, p)$，将其制作为时间矩阵 Q。

$$Q = \begin{pmatrix} 0 & \tilde{t}_{12\ min} & \tilde{t}_{13\ min} & \cdots & \tilde{t}_{1p\ min} \\ \tilde{t}_{21\ min} & 0 & \tilde{t}_{23\ min} & \cdots & \tilde{t}_{2p\ min} \\ \tilde{t}_{31\ min} & \tilde{t}_{32\ min} & 0 & \cdots & \tilde{t}_{3p\ min} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \tilde{t}_{p1\ min} & \tilde{t}_{p2\ min} & \tilde{t}_{p3\ min} & \cdots & 0 \end{pmatrix} \tag{33}$$

其中 $\tilde{t}_{kr}$ 的计算式为

$$\tilde{t}_{kr} = \sum_{j=1}^{m} \sum_{i^{(k)}=1}^{n} \left[\sum_{i^{(r)}=1}^{n} \left(x_{ki^{(k)}j} \ \cdot \ x_{ri^{(r)}j} \right) \ \cdot \ t'_{ri^{(k)}i^{(r)}} \right], \ k, r = 1, 2, \ldots, p \tag{34}$$

我们对决策变量矩阵 Y 与切换时间矩阵 Q 求 Hadamard 积总和，即可得到在包装印刷顺序不定的情形下的总切换时间 $\tilde{T}_{total}$。

$$\tilde{T}_{total} = \sum_{k=1}^{p} \sum_{r=1}^{p} Y \ \oplus \ Q \tag{35}$$

即

$$\tilde{T}_{total} = \sum_{k=1}^{p} \sum_{r=1}^{p} \left[y_{kr} \cdot \sum_{j=1}^{m} \sum_{i^{(k)}=1}^{n} \left(\sum_{i^{(r)}=1}^{n} \left(x_{ki^{(k)}j} \ \cdot \ x_{ri^{(r)}j} \right) \ \cdot \ t'_{ri^{(k)}i^{(r)}} \right) \right] \tag{36}$$

6.2 模型建立

添加上述修改后的目标函数，即可得到包装印刷顺序不定的情形下求解最小化切换时间的模型为

$$x_{kij} = \begin{cases} 1, & \text{打印包装 } k \text{ 时，插槽 } j \text{ 内插入墨盒 } i \\ 0, & \text{否则} \end{cases} \tag{37}$$

$$y_{kr} = \begin{cases} 1, & \text{由包装 } k \text{ 切换至包装 } r \\ 0, & \text{否则} \end{cases}, \ k, r = 1, 2, \ldots, p \tag{38}$$

$$\text{def} \quad C_k = \{i \mid i \text{ 为印刷包装 } k \text{ 所需要的墨盒}\} \tag{39}$$

$$\text{def} \quad \tilde{n} = \dim(C_k) \tag{40}$$

$$\min \quad \tilde{T}_{total} = \sum_{k=1}^{p}\sum_{r=1}^{p}\left[y_{kr} \cdot \sum_{j=1}^{m}\sum_{i^{(k)}=1}^{n}\left(\sum_{i^{(r)}=1}^{n}\left(x_{ki^{(k)}j} \cdot x_{ri^{(r)}j}\right) \cdot t'_{ri^{(k)}i^{(r)}}\right)\right] \tag{41}$$

$$\text{s.t.} \begin{cases} \sum\limits_{j=1}^{m} x_{kij} = 1, \ \ k = 1, 2, \ldots, p; \ i = 1, 2, \ldots, n, \\ \sum\limits_{i=1}^{n} x_{kij} \le 1, \ \ k = 1, 2, \ldots, p; \ j = 1, 2, \ldots, m \\ \sum\limits_{j=1}^{m} x_{ki'j} = 1, \ \ i' \in C_k; \ k = 1, 2, \ldots, p \\ \sum\limits_{j=1}^{m}\left(x_{ki_qj} \cdot j\right) < \sum\limits_{j=1}^{m}\left(x_{ki_{q+1}j} \cdot j\right), \ \ i_q \in C_k; \ q = 1, 2, \ldots, \tilde{n} - 1 \\ \sum\limits_{r=1}^{p} y_{kr} = 1, \ \ k = 1, 2, \ldots, p, \ k \ne k_{start} \\ \sum\limits_{k=1}^{p} y_{kr} = 1, \ \ r = 1, 2, \ldots, p, \ r \ne r_{end} \\ y_{kk} = 0, \ \ k = 1, 2, \ldots, p \\ x_{kij} = 0, 1; \ \ y_{kr} = 0, 1 \end{cases} \tag{42}$$

这是一个非线性 0-1 整数规划。

6.3 模型求解

6.3.1 基于 TSP 问题的求解方法

由于该模型中决策变量较为庞大，且目标函数具有较高的次数，直接使用 Gurobi 求解器尝试求解模型精确值的方法并不现实。因此，我们设计了如下算法，其能够计算出一个相对优秀的满意解。算法流程如下：

(1) 制作包装节点 B_k 的全连接带权有向图

基于计算得到的包装切换时间矩阵 Q，制作包装节点 B_k $(k = 1, 2 \ldots, p)$ 的全连接带权有向图。其中每条弧的权重即为矩阵 Q 中的最小切换时间 $\tilde{t}_{kr}$ $(k, r = 1, 2 \ldots, p)$。如下图所示：

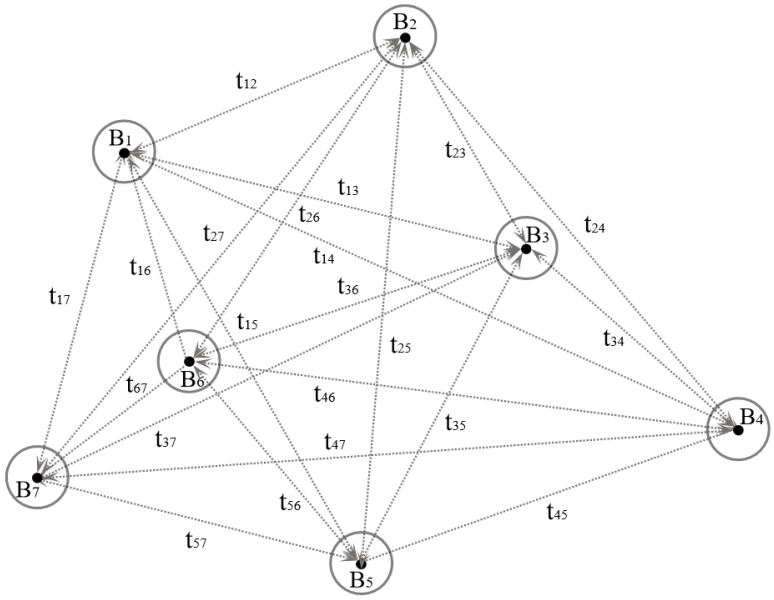


图 8: B_k 的全连接带权有向图

在得到的全连接带权有向图中，包装 B_k 的最优印刷顺序即是一个经过所有节点的有向连通路径，且需要使得该路径的长度最短。这使得我们可以将原本的最优排序问题转化为一个 TSP 问题的变体进行求解[3]。

基于定义的切换矩阵 Y 我们可以建立如下的 TSP 问题模型：

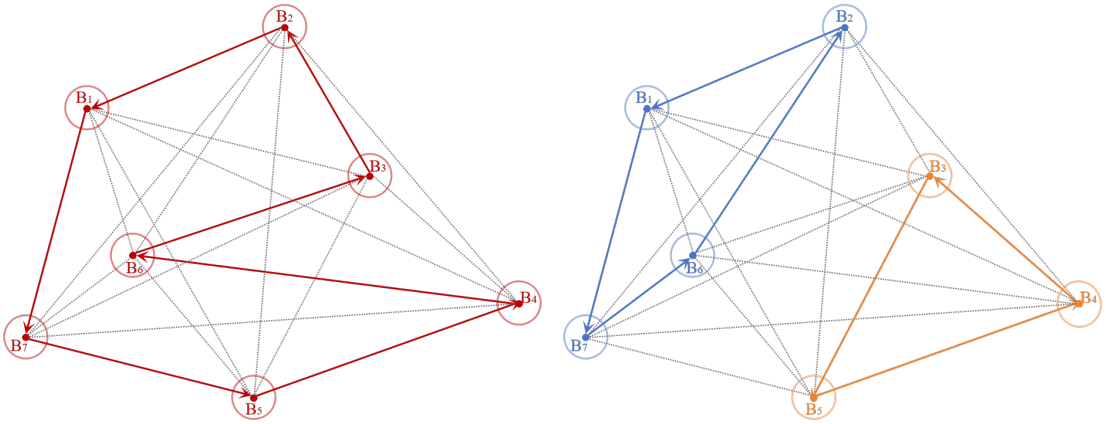
$$y_{kr} = \begin{cases} 1, & \text{路径经过弧 } (B_k, B_r) \\ 0, & \text{否则} \end{cases}, \ k, r = 1, 2, \ldots, p \tag{43}$$

$$\min \quad L = \sum_{k=1}^{p} \sum_{r=1}^{p} y_{kr} \cdot \tilde{t}_{kr\ min} \tag{44}$$

$$\text{s.t.} \begin{cases} \sum\limits_{r=1}^{p} y_{kr} = 1, \ \ k = 1, 2, \ldots, p \\ \sum\limits_{k=1}^{p} y_{kr} = 1, \ \ r = 1, 2, \ldots, p \\ y_{kk} = 0, \ \ k = 1, 2, \ldots, p \\ u_k - u_r + p y_{kr} \leq p - 1, \ 1 < k \neq j \leq p \end{cases} \tag{45}$$

模型中使用了 MTZ 方法来消除可能的子回路。MTZ 方法引入了一个新的标签变量 u 来标记每个节点，并通过确保在遍历过程中节点的 u 值保持增长趋势避免子回路的产生[4, 5]。

如果路径经过弧 (B_k, B_r)（即 $y_{kr} = 1$），则要求 $u_k < ur$。如果存在子回路，那么在这个子回路中，至少会有一对顶点或节点被重复访问，这将导致它们的 u 值在数值关系上产生冲突，从而证明该路径包含子回路。



(a) 无子回路的情况　　(b) 有子回路的情况

图 9: 子回路

与 DFJ 方法相比，MTZ 方法在处理较大规模问题时具有更高的效率，这使得我们的模型具有更高的求解速度。

(2) 求解 TSP 问题

由于模型中的节点个数较少，因此我们可以直接使用 Gurobi 求解器对该 TSP 模型进行求解。

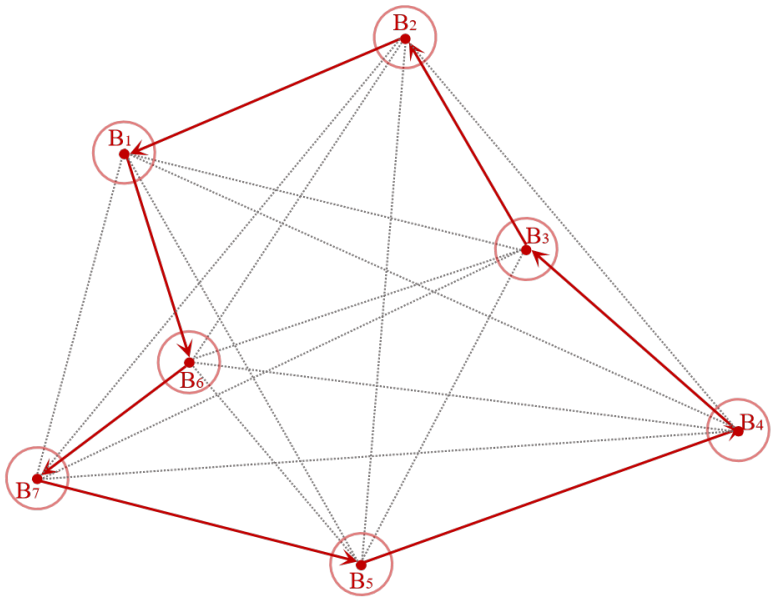


图 10: 求解结果

(3) 删除回路中的最长弧

得到最终包装印刷顺序前的最后一步是对 TSP 求解得到的最小有向回路进行破圈。由于每种包装的印刷都应在同一批内完成，因此形成回路的结果是不合理的。我们将回路中最长的一段有向弧删去，得到一个最小的单向路径。

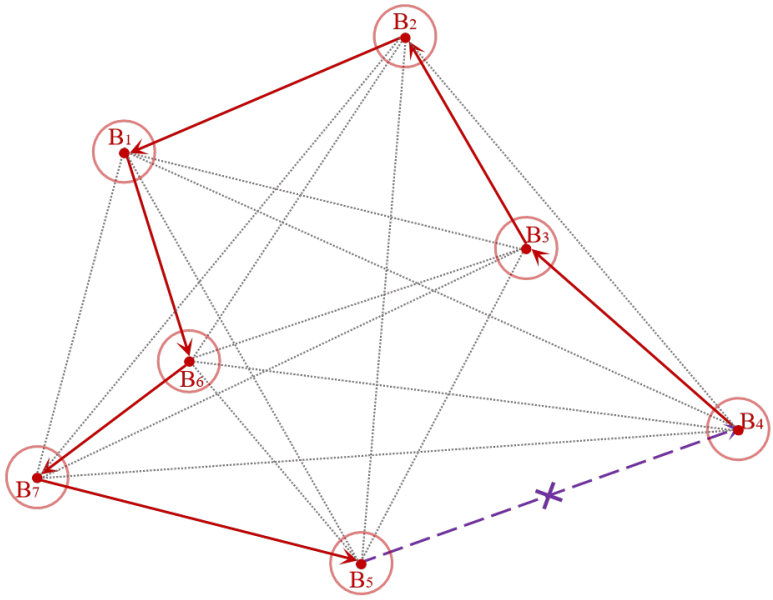


图 11: 删除最长弧

(4) 得到结果

将删去弧段的后继节点作为起始点，将其前驱节点作为终止点，沿路径遍历所有节点，我们可以得到最小化切换时间的包装种类 B_k 的印刷顺序。

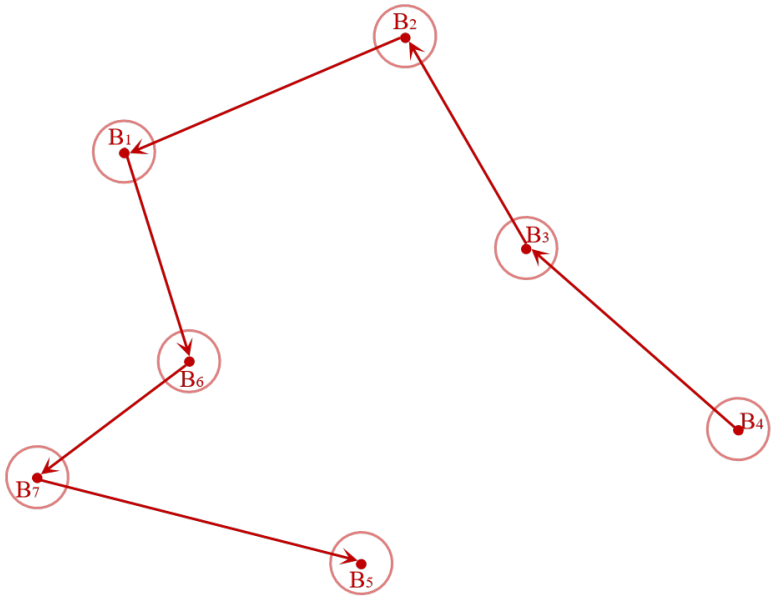


图 12: 包装印刷顺序结果

如上图所示的最佳包装印刷顺序即为：$B_4 \to B_3 \to B_2 \to B_1 \to B_6 \to B_7 \to B_5$。

上述方法的优势在于将原本的高阶优化问题转化为组合优化的形式，通过降低决策变量的维度来降低模型的时间复杂度，提高模型的求解效率；但其缺点在于模型的最优值不一定存在于最小回路中，且当变量规模较大时 TSP 问题无法求解出精确解，需要借助其他的启发式算法求解近似最优解。

6.3.2 模拟退火算法

对于变量较为庞大的情况（如样例 4），我们可以用模拟退火算法进行求解[6]。模拟退火算法是一种基于概率的随机寻优算法，它包含两个部分，即 Metropolis 算法和退火过程，分别对应模型的内循环和外循环。

Metropolis 算法是内循环，但在模型中我们对 Metropolis 算法进行了改变。在内循环中，我们先将包装种类先进行随机排序，得到一组随机的包装顺序向量

$$\tilde{B} = (B_1, B_2, \ldots, B_p) \tag{46}$$

在 $\tilde{B}$ 的排序结果上将墨盒 i 随机地插入插槽 j，通过一个三维数组 $X^{p\times n\times m}$ 来记录当前包装顺序下插槽中墨盒的排列情况（这个排列是随机的），在排序过程中保持墨盒的相对顺序不变。在这个三维数组 $X^{p\times n\times m}$ 中，p 轴上每一个二维数组 $X_k^{n\times m}$ $(k = 1, 2, \ldots, p)$ 即是一个随机生成的决策变量[7, 8]。

随后我们进行如下操作：

(1) 内循环

对于决策变量矩阵 X_k 求其与下一个矩阵 X_{k+1} 转置的乘积，得到一个 $n \times n$ 的墨盒切换数组 $S_{k,k+1}$。

$$S_{k,k+1} = X_k \cdot X_{k+1}^T \tag{47}$$

求出切换数组 $S_{k,k+1}$ 墨盒切换时间数组 T'_{k+1} 的 Hadamard 积总和，得到包装 k 与 $k+1$ 之间的墨盒切换时间 $\bar{t}_{k,k+1}$。

$$\bar{t}_{k,k+1} = \sum_{i=1}^{n}\sum_{j=1}^{m} S_{k,k+1} \oplus T'_{k+1} \tag{48}$$

对所有 $\tilde{t}$ 求和得到本次内循环的切换时间解 $\tilde{T}$。

$$\tilde{T} = \sum_{k=1}^{p-1} \tilde{t}_{k,k+1} \tag{49}$$

在内循环中不断求切换时间解，并将每个新获得的切换时间解 $\tilde{T}$ 与之前记录的最优解 $\tilde{T}_{\min}$ 进行对比，如果新的切换时间解更小，则将最优解赋值为新的切换时间解。并且记录下最优解的包装序列 $\tilde{B}$。

(2) 外循环

外循环不断降温进行内循环，最终在当前温度 T 达到或者小于终止温度 T_f 时完成求解，得到近似最优解 $\tilde{T}_{\min}$。

6.4 求解结果

从附件 4 中得到待求解模型的样例数据，各样例的模型参数如下表所示：

表 9: 模型四相关数据参数

实例	Ins1	Ins2	Ins3	Ins4
包装种类数 p	5	5	7	20
墨盒数 n	10	10	10	40
插槽数 m	2	3	2	10

将相应数据带入模型中，使用 Gurobi 求解器及模拟退火算法求出最优解。

表 10: 墨盒切换最短时间

实例	Ins1	Ins2	Ins3	Ins4
T_{total}	33	63	92	767

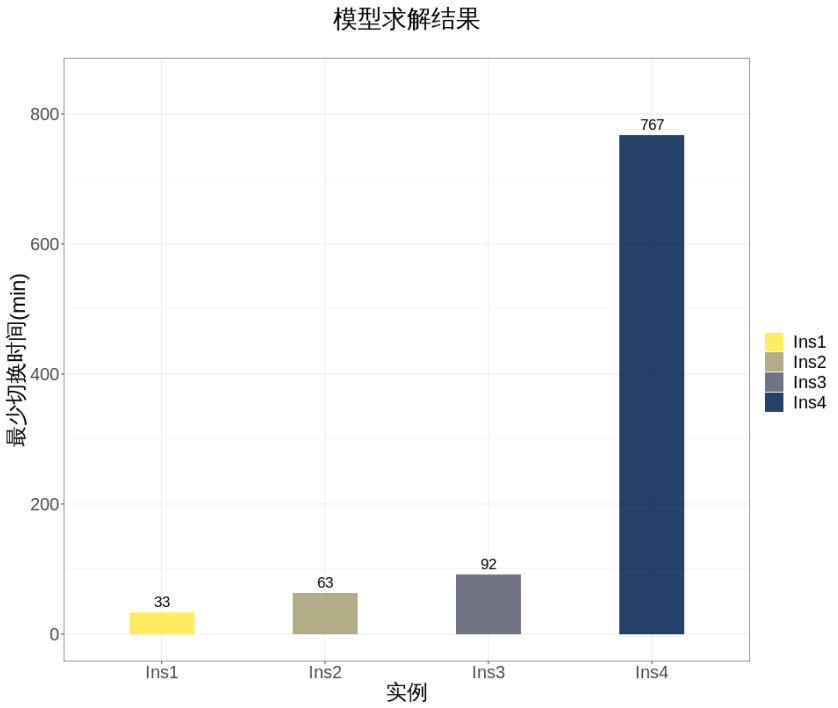


图 13: 模型求解结果

参考文献

[1] 李映红. 线性 0—1 规划模型的排序解法 [J]. 西南交通大学学报,2001,36(5):468-.

[2] 薄桂华, 黄敏, 王洪峰.4PL 路径优化问题 0-1 规划模型与求解 [J]. 控制工程,2013,20(2):239-242.

[3] 许志聪.TSP 问题及其解法研究 [J]. 大众科技,2008(10):50-51.

[4] Martin, Desrochers., Gilbert, Laporte. (1991). Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. Operations Research Letters, 10(1), 27-36. Available from: 10.1016/0167-6377(91)90083-2

[5] Moustapha, Diaby. (2006). The traveling salesman problem: A Linear programming formulation. arXiv: Computational Complexity.

[6] 包强. 一种求解旅行商问题的混合遗传模拟退火算法 [J]. 中国储运,2021(11):204-205.

[7] 谭宁波, 王典华, 熊仁刚. 扩展旅行商问题模型研究 [J]. 内江师范学院学报,2009,24(B07):262-263.

[8] 齐安智. 一种基于改进模拟退火算法的 TSP 问题的应用研究 [J]. 信息与电脑,2020,32(3):32-34.

A 附录 A：源代码

运行代码须申请 Gurobi 使用权限，额外安装 Python 的 gurobipy，pandas，numpy，matplotlib，ast 库，并将附件文件包括其中文件放在程序父文件夹同级目录中。

代码 1: 问题一求解

```
import numpy as np
import pandas as pd
import gurobipy as grb
from gurobipy import *
from time import sleep


# 读取文件数据
# 读取的文件名
def selectFile(n, m):
    filepath = f"..\\附件{n}\\"
    filenames = ["Ins1_5_10_2.xlsx", "Ins2_7_10_3.xlsx",
        "Ins3_10_50_15.xlsx", "Ins4_20_50_10.xlsx",
            "Ins5_30_60_10.xlsx"]
    filename = filepath + filenames[m]
    return filename


def solveModel(filename, M):
    # 获取模型中墨盒、插槽及包装的数量
    information = pd.read_excel(filename, sheet_name="包装-墨盒-插槽")
    # 设定三维数组的大小(颜色/插槽/包装)
    color = information['墨盒编号'].notnull().sum()
    slot = information['插槽序号'].notnull().sum()
    package = information['包装种类编号'].notnull().sum()

    # 获取每种包装所需的墨盒编号
    color_need = pd.read_excel(filename, sheet_name="包装种类及其所需墨盒")
    # 使用apply函数将字符串转换为整数元组，并存储在color_need中
    color_need = color_need['所需墨盒编号'].apply(
        lambda x: tuple(int(num.strip()) for num in
            x.strip('[]').split(',')))

    # 创建更换时间矩阵(模型中只需求更换次数，因此除对角线外均为1)
    change_time = np.ones((package, color, color), dtype=int)
```

```
34          # 使用大M法简化时间矩阵
35          for k in range(package):
36              for i in range(color):
37                  if i + 1 not in color_need[k]:
38                      change_time[k, :, i] = M
39                  change_time[k, i, i] = 0 # 将对角线设置为0
40
41          # 创建决策变量
42          # 创建一个新的模型
43          model = grb.Model('Printer Exchange Colors')
44
45          # 创建一个三维字典来存储决策变量
46          variables = {}
47          for k in range(package):
48              for i in range(color):
49                  for j in range(slot):
50                      # 添加变量 x_kij
51                      variables[k, i, j] = model.addVar(vtype=grb.GRB.BINARY,
                            name=f"x_{k}{i}{j}")
52
53          # 添加约束
54          # 对于特定的包装，印刷过程中每个插槽只能有一个墨盒
55          for package_index in range(package):
56              for slot_index in range(slot):
57                  #
                        使用列表推导式创建一个生成器表达式，它包含了slot列中所有决策变量的和
58                  slot_sum = grb.quicksum(variables[package_index, color,
                        slot_index] for color in range(color))
59                  # 约束每个变量和=1
60                  model.addConstr(slot_sum == 1,
                        name=f"slot_sum_{slot_index}_in_{package_index}")
61
62          # 对于特定的包装，印刷过程中一个墨盒最多插在一个插槽内
63          for package_index in range(package):
64              for color_index in range(color):
65                  #
                        使用列表推导式创建一个生成器表达式，它包含了color行中所有决策变量的和
66                  color_sum = grb.quicksum(variables[package_index, color_index,
                        slot] for slot in range(slot))
67                  # 约束每个变量和<=1
68                  model.addConstr(color_sum <= 1,
```

```
                name=f"color_sum_{color_index}_in_{package_index}")

        # 对于特定的包装，需要保证有相应颜色的墨盒
        for package_index in range(package):
            for color_index in color_need[package_index]:
                target_color = grb.quicksum(variables[package_index, color_index
                    - 1, slot] for slot in range(slot))
                # 约束每个变量和=1
                model.addConstr(target_color == 1,
                    name=f"target_color_{color_index}_in_{package_index}")

        # 设置目标函数
        total_time = grb.quicksum(variables[k, i, j] * variables[k + 1, ii, j]
            * change_time[k + 1, i, ii]
                        for k in range(package - 1) for j in range(slot)
                        for i in range(color) for ii in range(color))

        model.setObjective(total_time, sense=grb.GRB.MINIMIZE)
        model.setParam('Presolve', 2)

        # 求解模型并输出结果
        # 求解模型
        model.optimize()

        # 输出结果
        print('Optimal objective: %g' % model.objVal)
        for k in range(package):
            for i in range(color):
                for j in range(slot):
                    print(f'x_{k + 1}_{i + 1}_{j + 1}: {variables[k, i, j].x}',
                        end=' ')
                print()
            print()

    if __name__ == '__main__':
        M = 10
        for m in range(5):
            filename = selectFile(1, m)
            solveModel(filename, M)
            sleep(10)
```

代码 2: 问题三求解

```
    import numpy as np
import pandas as pd
import gurobipy as grb
from gurobipy import *
from time import sleep


# 读取文件数据
# 读取的文件名
def selectFile(n, m):
    filepath = f"..\\附件{n}\\"
    filenames = ["Ins1_5_10_3.xlsx", "Ins2_7_10_2.xlsx", "Ins3_10_30_10.xlsx",
        "Ins4_20_40_10.xlsx",
            "Ins5_30_60_10.xlsx"]
    filename = filepath + filenames[m]
    return filename



def solveModel(filename, M, m):
    # 获取模型中墨盒、插槽及包装的数量
    information = pd.read_excel(filename, sheet_name="包装-墨盒-插槽")
    # 设定三维数组的大小(颜色/插槽/包装)
    color = information['墨盒编号'].notnull().sum()
    slot = information['插槽序号'].notnull().sum()
    package = information['包装种类编号'].notnull().sum()

    # 获取每种包装所需的墨盒编号
    color_need = pd.read_excel(filename, sheet_name="包装种类及其所需墨盒")
    # 使用apply函数将字符串转换为整数元组，并存储在color_need中
    color_need = color_need['所需墨盒编号'].apply(
        lambda x: tuple(int(num.strip()) for num in x.strip('[]').split(',')))

    # 创建更换时间矩阵(模型中只需求更换次数，因此除对角线外均为1)
    time_information = pd.read_excel(filename, sheet_name="墨盒切换时间")
    time_information.drop(time_information.columns[0], axis=1, inplace=True)
    change_time = np.array(time_information)
    change_time = np.stack([change_time] * package, axis=0)
    # 使用大M法简化时间矩阵
    for k in range(package):
```

```
        for i in range(color):
            if i + 1 not in color_need[k]:
                change_time[k, :, i] = M
            change_time[k, i, i] = 0 # 将对角线设置为0

    # 创建决策变量
    # 创建一个新的模型
    model = grb.Model('Printer Exchange Colors')
    if m > 1:
        # 结果精度
        # model.Params.MIPGap = 28*10**(-2)
        # 增加启发式的使用，可以提高找到可行解的机会。值的范围是0到1，默认值是0.05
        model.Params.Heuristics = 0.5

    # 创建一个三维字典来存储决策变量
    variables = {}
    for k in range(package):
        for i in range(color):
            for j in range(slot):
                # 添加变量 x_kij
                variables[k, i, j] = model.addVar(vtype=grb.GRB.BINARY,
                    name=f"x_{k}{i}{j}")

    # 添加约束
    # 对于特定的包装，印刷过程中每个插槽只能有一个墨盒
    for package_index in range(package):
        for slot_index in range(slot):
            # 使用列表推导式创建一个生成器表达式，它包含了slot列中所有决策变量的和
            slot_sum = grb.quicksum(variables[package_index, color, slot_index]
                for color in range(color))
            # 约束每个变量和=1
            model.addConstr(slot_sum == 1,
                name=f"slot_sum_{slot_index}_in_{package_index}")

    # 对于特定的包装，印刷过程中一个墨盒最多插在一个插槽内
    for package_index in range(package):
        for color_index in range(color):
            # 使用列表推导式创建一个生成器表达式，它包含了color行中所有决策变量的和
            color_sum = grb.quicksum(variables[package_index, color_index,
                slot] for slot in range(slot))
            # 约束每个变量和<=1
```

```
77            model.addConstr(color_sum <= 1,
                  name=f"color_sum_{color_index}_in_{package_index}")
78
79        # 对于特定的包装，需要保证有相应颜色的墨盒
80        for package_index in range(package):
81            for color_index in color_need[package_index]:
82                target_color = grb.quicksum(variables[package_index, color_index -
                      1, slot] for slot in range(slot))
83                # 约束每个变量和=1
84                model.addConstr(target_color == 1,
                      name=f"target_color_{color_index}_in_{package_index}")
85
86        # 设置目标函数
87        total_time = grb.quicksum(variables[k, i, j] * variables[k + 1, ii, j] *
              change_time[k + 1, i, ii]
88                                  for k in range(package - 1) for j in range(slot)
89                                  for i in range(color) for ii in range(color))
90
91        model.setObjective(total_time, sense=grb.GRB.MINIMIZE)
92        model.setParam('Presolve', 2)
93
94        # 求解模型并输出结果
95        # 求解模型
96        model.optimize()
97
98        # 输出结果
99        print('Optimal objective: %g' % model.objVal)
100       for k in range(package):
101           for i in range(color):
102               for j in range(slot):
103                   print(f'x_{k + 1}_{i + 1}_{j + 1}: {abs(variables[k, i, j].x)}',
                          end=' ')
104               print()
105           print()
106
107   if __name__ == '__main__':
108       M = 1000
109       for m in range(5):
110           filename = selectFile(2, m)
111           solveModel(filename, M, m)
112           sleep(10)
```

代码 3: 问题三求解

```
import numpy as np
import pandas as pd
import gurobipy as grb
from gurobipy import *
from time import sleep


# 读取文件数据
# 读取的文件名
def selectFile(n, m):
    filepath = f"..\\附件{n}\\"
    filenames = ["Ins1_5_5_2.xlsx", "Ins2_10_30_10.xlsx",
        "Ins3_20_50_10.xlsx", "Ins4_30_60_10.xlsx"]
    filename = filepath + filenames[m]
    return filename


def solveModel(filename, M, m):
    # 获取模型中墨盒、插槽及包装的数量
    information = pd.read_excel(filename, sheet_name="包装-墨盒-插槽")
    # 设定三维数组的大小(颜色/插槽/包装)
    color = information['墨盒编号'].notnull().sum()
    slot = information['插槽序号'].notnull().sum()
    package = information['包装种类编号'].notnull().sum()

    # 获取每种包装所需的墨盒编号
    color_need = pd.read_excel(filename, sheet_name="包装种类及其所需墨盒")
    # 使用apply函数将字符串转换为整数元组，并存储在color_need中
    color_need = color_need['所需墨盒编号'].apply(lambda x:
        tuple(int(num.strip()) for num in x.strip('[]').split(',')))

    # 创建更换时间矩阵
    time_information = pd.read_excel(filename, sheet_name="墨盒切换时间")
    time_information.drop(time_information.columns[0], axis=1, inplace=True)
    change_time = np.array(time_information)
    change_time = np.stack([change_time] * package, axis=0)
    # 使用大M法简化时间矩阵
    for k in range(package):
        for i in range(color):
            if i + 1 not in color_need[k]:
```

```
39              change_time[k, :, i] = M
40          change_time[k, i, i] = 0 # 将对角线设置为0
41
42      # 创建决策变量
43      # 创建一个新的模型
44      model = grb.Model('Printer Exchange Colors')
45      if m > 1:
46          # 结果精度
47          # model.Params.MIPGap = 28*10**(-2)
48          # 增加启发式的使用，可以提高找到可行解的机会。值的范围是0到1，默认值是0.05
49          model.Params.Heuristics = 0.5
50
51      # 创建一个三维字典来存储决策变量
52      variables = {}
53      for k in range(package):
54          for i in range(color):
55              for j in range(slot):
56                  # 添加变量 x_kij
57                  variables[k, i, j] = model.addVar(vtype=grb.GRB.BINARY,
                        name=f"x_{k}{i}{j}")
58
59      # 添加约束
60      # 对于特定的包装，印刷过程中每个插槽只能有一个墨盒
61      for package_index in range(package):
62          for slot_index in range(slot):
63              # 使用列表推导式创建一个生成器表达式，它包含了slot列中所有决策变量的和
64              slot_sum = grb.quicksum(variables[package_index, color, slot_index]
                    for color in range(color))
65              # 约束每个变量和=1
66              model.addConstr(slot_sum == 1,
                    name=f"slot_sum_{slot_index}_in_{package_index}")
67
68      # 对于特定的包装，印刷过程中一个墨盒最多插在一个插槽内
69      for package_index in range(package):
70          for color_index in range(color):
71              # 使用列表推导式创建一个生成器表达式，它包含了color行中所有决策变量的和
72              color_sum = grb.quicksum(variables[package_index, color_index,
                    slot] for slot in range(slot))
73              # 约束每个变量和<=1
74              model.addConstr(color_sum <= 1,
                    name=f"color_sum_{color_index}_in_{package_index}")
```

```
75
76      # 对于特定的包装，需要保证有相应颜色的墨盒
77      for package_index in range(package):
78          for color_index in color_need[package_index]:
79              target_color = grb.quicksum(variables[package_index, color_index -
                    1, slot] for slot in range(slot))
80              # 约束每个变量和=1
81              model.addConstr(target_color == 1,
                    name=f"target_color_{color_index}_in_{package_index}")
82
83      # 固定墨盒装填顺序
84      slot_all_index = [i for i in range(1, slot + 1)]
85      for package_index in range(package):
86          index = 0
87          choice = list()
88          for color_index in color_need[package_index]:
89              choice.append(grb.quicksum(
90                  variables[package_index, color_index - 1, slot] *
                        slot_all_index[slot] for slot in range(slot)))
91              if index:
92                  model.addConstr(choice[index - 1] - choice[index] <= 0,
                        name=f"choice_{color_index}_in_{package_index}")
93
94              index += 1
95
96      # 设置目标函数
97      total_time = grb.quicksum(variables[k, i, j] * variables[k + 1, ii, j] *
            change_time[k + 1, i, ii]
98                        for k in range(package - 1) for j in range(slot)
99                        for i in range(color) for ii in range(color))
100
101     model.setObjective(total_time, sense=grb.GRB.MINIMIZE)
102     model.setParam('Presolve', 2)
103     # 求解模型并输出结果
104     # 求解模型
105     model.optimize()
106     # 输出结果
107     print('Optimal objective: %g' % model.objVal)
108     for k in range(package):
109         for i in range(color):
110             for j in range(slot):
```

```
111             print(f'x_{k + 1}_{i + 1}_{j + 1}: {abs(variables[k, i, j].x)}',
                    end=' ')
112         print()
113     print()
114
115 if __name__ == '__main__':
116     M = 1000
117     for m in range(4):
118         filename = selectFile(3, m)
119         solveModel(filename, M, m)
120         sleep(10)
```

代码 4: 问题三求解

```
1  import numpy as np
2  import pandas as pd
3  import gurobipy as grb
4  from gurobipy import *
5  from time import sleep
6
7
8  # 读取文件数据
9  # 读取的文件名
10 def selectFile(n, m):
11     filepath = f"..\\附件{n}\\"
12     filenames = ["Ins1_5_10_2.xlsx", "Ins2_5_10_3.xlsx", "Ins3_7_10_2.xlsx",
           "Ins4_20_40_10.xlsx"]
13     filename = filepath + filenames[m]
14     return filename
15
16
17 def solveModel(filename, M, m):
18     # 获取模型中墨盒、插槽及包装的数量
19     information = pd.read_excel(filename, sheet_name="包装-墨盒-插槽")
20     # 设定三维数组的大小(颜色/插槽/包装)
21     color = information['墨盒编号'].notnull().sum()
22     slot = information['插槽序号'].notnull().sum()
23     package = information['包装种类编号'].notnull().sum()
24
25     # 获取每种包装所需的墨盒编号
26     color_need = pd.read_excel(filename, sheet_name="包装种类及其所需墨盒")
```

```
    # 使用apply函数将字符串转换为整数元组，并存储在color_need中
    color_need = color_need['所需墨盒编号'].apply(lambda x:
        tuple(int(num.strip()) for num in x.strip('[]').split(',')))

    # 创建更换时间矩阵
    time_information = pd.read_excel(filename, sheet_name="墨盒切换时间")
    time_information.drop(time_information.columns[0], axis=1, inplace=True)
    change_time = np.array(time_information)
    change_time = np.stack([change_time] * package, axis=0)
    # 使用大M法简化时间矩阵
    for k in range(package):
        for i in range(color):
            if i + 1 not in color_need[k]:
                change_time[k, :, i] = M
            change_time[k, i, i] = 0 # 将对角线设置为0

    # 创建决策变量
    # 创建一个新的模型
    model = grb.Model('Printer Exchange Colors')
    if m > 1:
        # 结果精度
        # model.Params.MIPGap = 28*10**(-2)
        # 增加启发式的使用，可以提高找到可行解的机会。值的范围是0到1，默认值是0.05
        model.Params.Heuristics = 0.5

    # 创建一个三维字典来存储决策变量
    variables = {}
    for k in range(package):
        for i in range(color):
            for j in range(slot):
                # 添加变量 x_kij
                variables[k, i, j] = model.addVar(vtype=grb.GRB.BINARY,
                    name=f"x_{k}{i}{j}")

    # 二维包装顺序决策变量
    variables_2 = {}
    for i in range(package):
        for j in range(package):
            # 添加变量 y_ij
            variables_2[i, j] = model.addVar(vtype=grb.GRB.BINARY,
                name=f"y_{i}{j}")
```

```
65
66      # 添加约束
67      # 包装必须更换
68      exchange_no_sum = grb.quicksum(variables_2[package_index, package_index]
            for package_index in range(package))
69      model.addConstr(exchange_no_sum == 0,
            name=f"exchange_no_sum_{package}_{package}")
70
71      # 每种包装之后只有一种包装类型
72      for package_index in range(package):
73          package1_sum = grb.quicksum(variables_2[package_index, j] for j in
                range(package) if j != package_index)
74          # 变量和为1
75          model.addConstr(package1_sum == 1,
                name=f"package1_sum_{package_index}_to")
76
77      # 每种包装必须换一次墨盒
78      for package_index in range(package):
79          package2_sum = grb.quicksum(variables_2[i, package_index] for i in
                range(package) if i != package_index)
80          # 变量和为1
81          model.addConstr(package2_sum == 1,
                name=f"package2_sum_{package_index}_from")
82
83      # 对于特定的包装，印刷过程中每个插槽只能有一个墨盒
84      for package_index in range(package):
85          for slot_index in range(slot):
86              # 使用列表推导式创建一个生成器表达式，它包含了slot列中所有决策变量的和
87              slot_sum = grb.quicksum(variables[package_index, color, slot_index]
                    for color in range(color))
88              # 约束每个变量和=1
89              model.addConstr(slot_sum == 1,
                    name=f"slot_sum_{slot_index}_in_{package_index}")
90
91      # 对于特定的包装，印刷过程中一个墨盒最多插在一个插槽内
92      for package_index in range(package):
93          for color_index in range(color):
94              # 使用列表推导式创建一个生成器表达式，它包含了color行中所有决策变量的和
95              color_sum = grb.quicksum(variables[package_index, color_index,
                    slot] for slot in range(slot))
96              # 约束每个变量和<=1
```

```
97            model.addConstr(color_sum <= 1,
                  name=f"color_sum_{color_index}_in_{package_index}")
98
99        # 对于特定的包装，需要保证有相应颜色的墨盒
100       for package_index in range(package):
101          for color_index in color_need[package_index]:
102             target_color = grb.quicksum(variables[package_index, color_index -
                    1, slot] for slot in range(slot))
103             # 约束每个变量和=1
104             model.addConstr(target_color == 1,
                    name=f"target_color_{color_index}_in_{package_index}")
105
106       # 固定墨盒装填顺序
107       slot_all_index = [i for i in range(1, slot + 1)]
108       for package_index in range(package):
109          index = 0
110          choice = list()
111          for color_index in color_need[package_index]:
112             choice.append(grb.quicksum(
113                variables[package_index, color_index - 1, slot] *
                       slot_all_index[slot] for slot in range(slot)))
114             if index:
115                model.addConstr(choice[index - 1] - choice[index] <= 0,
                       name=f"choice_{color_index}_in_{package_index}")
116
117             index += 1
118
119       # 定义回调函数 消除子回路
120       def subtourelim(model, where):
121          if where == GRB.Callback.MIPSOL:
122             # 获取当前解
123             vals = model.cbGetSolution([model._variables_2[i, j] for i in
                    range(package) for j in range(package)])
124             # 创建一个空的子回路列表
125             selected = gurobipy.tuplelist()
126             # 遍历决策变量，找出值为1的变量，即路径
127             for i in range(package):
128                selected.extend((i, j) for j in range(package) if vals[i *
                       package + j] > 0.5)
129             # 寻找子回路
130             tours = []
```

```
131             while selected:
132                 # 开始新的子回路
133                 tour = [selected[0]]
134                 selected.remove(selected[0])
135                 while True:
136                     # 找到当前子回路的最后一个节点
137                     last = tour[-1][1]
138                     # 检查是否有从最后一个节点出发的路径
139                     next_edge = selected.select(last, '*')
140                     if not next_edge:
141                         break
142                     # 添加下一条边到子回路
143                     tour.append(next_edge[0])
144                     selected.remove(next_edge[0])
145                 # 添加子回路到列表
146                 tours.append(tour)
147             # 如果找到的子回路数量大于1，则添加约束以消除子回路
148             for tour in tours:
149                 if len(tour) < package:
150                     model.cbLazy(quicksum(model._variables_2[i, j] for i, j in
                            tour) <= len(tour) - 1)
151
152     # 将决策变量传递给回调函数
153     model._variables_2 = variables_2
154
155     # 设置使用lazy constraints
156     model.Params.lazyConstraints = 1
157
158     # 设置目标函数
159     # 创建一个新的变量表示每个包装的切换次数
160     switch_count = {}
161     for i in range(package):
162         for j in range(package):
163             # 添加变量 switch_count
164             switch_count[i, j] = model.addVar(vtype=grb.GRB.INTEGER,
                        name="switch_count")
165
166     # 约束每个包装的切换次数
167     for k in range(package):
168         for kk in range(package):
169             if kk != k:
```

```
            model.addConstr(
                switch_count[k, kk] == grb.quicksum(
                    variables[k, i, j] * variables[kk, ii, j] * change_time[kk,
                        i, ii]
                    for i in range(color)
                    for ii in range(color)
                    for j in range(slot)),
                name=f"switch_count_{k}_{kk}")

    # 计算总时间
    total_time = grb.quicksum(variables_2[k, kk] * switch_count[k, kk]
                      for k in range(package)
                      for kk in range(package) if kk != k)

    model.setObjective(total_time, sense=grb.GRB.MINIMIZE)
    model.setParam('Presolve', 2)
    # 求解模型并输出结果
    # 求解模型 回调函数subtourelim
    model.optimize(subtourelim)
    # 输出结果
    for i in range(package):
        for j in range(package):
            print(f'y_{i + 1}_{j + 1}: {abs(variables_2[i, j].x)}', end='\t')
        print()
    print()

    for i in range(package):
        for j in range(package):
            print(f'switch_count_{i + 1}_{j + 1}: {abs(switch_count[i, j].x)}',
                  end='\t')
        print()
    print()

    print('Optimal objective: %g' % model.objVal)
    for k in range(package):
        for i in range(color):
            for j in range(slot):
                print(f'x_{k + 1}_{i + 1}_{j + 1}: {abs(variables[k, i, j].x)}',
                      end='\t')
            print()
        print()
```

```
208
209         # 进一步运算
210         max = 0
211         index = [0, 0]
212         step = 0
213         for i in (variables_2[k, kk].x * switch_count[k, kk].x for k in
                range(package) for kk in range(package)):
214             if i and i > max:
215                 max = i
216                 index[0] = step // package
217                 index[1] = step % package
218             step += 1
219
220         print('Optimal objective update: %g' % (model.objVal - max))
221         print(f'set y_{index[0] + 1}_{index[1] + 1}: {0}', end='\n')
222
223         # 打印包装顺序
224         path = []
225         start = index[1]
226         print("包装打印顺序: ", start + 1, end='')
227         i = 0
228         while 1:
229             if variables_2[start, i].x:
230                 if i == index[1]:
231                     break
232                 print(end='->')
233                 path.append(i)
234                 print(i + 1, end='')
235                 start = i
236                 i = 0
237                 continue
238             i += 1
239         print('\n')
240
241     if __name__ == '__main__':
242         M = 1000
243         for m in range(4):
244             filename = selectFile(4, m)
245             solveModel(filename, M, m)
246             sleep(10)
```

代码 5: 求解并保存文件

```
import subprocess
import time


# 定义运行脚本并保存输出的函数
def run_script(script_name, output_file):
    with open(output_file, 'w') as file:
        # 运行脚本并将输出重定向到文件
        subprocess.run(['python', script_name], stdout=file, text=True)


# 按顺序运行代码程序
run_script('model1.py', 'output1_1.txt')
time.sleep(10)
run_script('model2.py', 'output2_1.txt')
time.sleep(10)
run_script('model3.py', 'output3_1.txt')
time.sleep(10)
run_script('model4.py', 'output4_1.txt')
```

代码 6: 模拟退火算法

```
import random
import ast as at
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt


#excel文件路径
file_path = 'Ins4_20_40_10.xlsx'

# 访问指定工作表
sheet1 = '包装-墨盒-插槽'
sheet2 = '包装种类及其所需墨盒'
sheet3 = '墨盒切换时间'

#读取指定工作表的内容
data1 = pd.read_excel(file_path, sheet_name = sheet1)
```

```
19 data2 = pd.read_excel(file_path, sheet_name = sheet2)
20 data3 = pd.read_excel(file_path, sheet_name = sheet3)
21
22 #化为元组取长度和所需颜色
23 color_need = data2['所需墨盒编号'].apply(lambda x: tuple(int(num.strip()) for
       num in x.strip('[]').split(',')))
24 print(color_need[0])
25
26 #转换为数组
27 data1 = data1.to_numpy()
28 data2 = data2.to_numpy()
29 data3 = data3.to_numpy()
30
31 def delete_column(arr, col_index):
32     return np.delete(arr, col_index, axis=1)
33
34 col_index = 0 # 要删除的列索引
35 new_data3 = delete_column(data3, col_index)
36
37 #print(data1)
38 #print(data2)
39 #print(new_data3)
40
41 # 使用numpy的isnan()函数检查数组中的每个元素是否为nan
42 num_package = np.isnan(data1[:, 0])#包装
43 num_color = np.isnan(data1[:, 1])#墨盒
44 num_slot = np.isnan(data1[:, 2])#插槽
45
46 # 使用numpy的sum()函数计算非nan值的数量
47 non_num_package = np.sum(~num_package)
48 non_nan_color = np.sum(~num_color)
49 non_nan_slot = np.sum(~num_slot)
50
51 print(non_num_package, non_nan_color, non_nan_slot)
52
53
54 history = {'array_T': [], 'array_time': []}
55
56 iter = 100 #内循环迭代次数
57 alpha = 0.99 #降温系数
58 T0 = 100 #初始温度
```

```
T = T0 #当前温度
Tf = 0.01 #终值温度
best_time = 10000 #设置一个大的时间初始值

while(T > Tf):

    for p in range(iter):
        #建立一个全0三维数组
        x1 = np.zeros((non_nan_color, non_nan_slot, non_num_package))
        # 随机排列数组的行
        np.random.shuffle(color_need)

        #按照包装顺序将墨盒按顺序要求随机插入插槽，再填至x1数组中，x轴为墨盒，y轴为插槽，z轴为包装
        for k in range(non_num_package):
            #将空列表来存储当前包装墨盒所插插槽
            random_arr = []
            #设置信号，当信号为0时满足条件
            flag = 1
            #将墨盒插槽不断进行循环生成排序，直到满足条件
            while(flag):
                i = 0
                random_arr = np.random.choice(range(0, non_nan_slot),
                    size=len(color_need[k]), replace=False)
                while(random_arr[i] < random_arr[i + 1]):
                    i += 1
                    if (i == len(color_need[k]) - 1):
                        flag = 0
                        break

            #print(random_arr)

            #将墨盒插入的插槽赋值为1
            for i, j in zip(color_need[k], random_arr):
                x1[i-1][j][k] = 1

        #print(x1[:, :, 0])

        #将切换的墨盒取到记录，同时记录切换时间
        while(1):
            Time_sum = 0

```

```
            Time = []
            X = []
            indexes = []
            change_x = []
            #取得对应的插槽切换，并且计算出时间
            X = np.dot(x1[:, :, 0], np.transpose(x1[:, :, 1]))
            change_x = x1[:, :, 0] + x1[:, :, 1]
            indexes = np.argwhere(X == 1)
            Time = X * new_data3
            #print(indexes)

            for m in range(non_num_package - 2):

                X = np.dot(change_x, np.transpose(x1[:, :, m+2]))
                change_x = change_x + x1[:, :, m+2]
                indexes = np.argwhere(X == 1)
                if np.sum(indexes) != 0:
                    for q in indexes[0, :]:
                        change_x[q] = 0
                else:
                    continue
                Time += X * new_data3

            Time_sum = np.sum(Time)

            if (Time_sum < best_time):
                best_time = Time_sum
            break
        print(best_time)
    # 迭代p次记录在该温度下最优解
    history['array_time'].append(best_time)
    history['array_T'].append(T)
    T = T * alpha

plt.plot(history['array_T'], history['array_time'])
plt.title('Tendency')
plt.xlabel('T')
plt.ylabel('time')
plt.gca().invert_xaxis()
plt.show()
print(best_time)
```