基于整数规划的柔性版印刷机墨盒切换策略

摘　要

柔性版印刷是一种绿色印刷方式，逐渐占据了我国印刷行业的主导地位。该方式对于某种包装的印刷，需要多种颜色的墨盒协调配合来完成。柔性版印刷的插槽数量是固定的，因此柔性版印刷的时间极大地依赖于墨盒的切换时间。本文在上述背景下，将墨盒序列切换抽象为关于**整数规划**的数学问题，从墨盒总切换时间以及切换次数进行替换策略的研究。

问题一要求本文在只有一台喷雾清洗机且确定包装种类印刷顺序的前提下，研究出一种墨盒的替换策略使得切换次数最小。本文通过将每种包装的印刷过程抽象为墨盒之间的切换过程，运用 **0-1 规划**的思想，将机器上的墨盒序列作为决策变量，设定总切换次数为目标函数求解模型。针对附件 1 的多个数据，本文求得切换方案最少分别需要 **5，8，48，58，130** 次。

问题二在问题一的基础上将切换的总时间的最小值作为解决目标。因此本文在问题一求解模型的基础上更改了求解目标，针对附件 2 的多个数据，求得切换方案最少分别需要 **32，51，111，203，326** 秒。

问题三在问题一的基础上更改了题目条件，限制了包装印刷时所需要的墨盒的印刷顺序，因此在模型二的基础上添加了新的约束，约束了印刷机上墨盒的摆放顺序与包装的颜色顺序严格相同。针对附件 3 的多个数据求得切换方案最少分别需要 **56，184，245，288** 秒。

问题四在问题三的基础上进一步修改条件，每轮印刷的包装顺序可以改变。本文在前面建立的模型基础上，进行适当地修改，每轮打印时将所有的包装作为潜在目标，为每轮选择一个合适的包装印刷，并求解最小替换时间。针对附件 4 的多个数据，求得切换方案最少分别需要 **20，37，65，196** 秒。

问题五在问题四的基础上添加了一台清洗机，使得在每一伦次的印刷结束后可以双线程地清洗，这大大缩减了替换所需时间。本文采用**最小值最大化的策略**，来最小化两组清洗时间的最大值来二分每一轮清洗分为时间尽可能相等的两组，利用求解器求得一个最佳策略使得整个印刷过程的切换时间最小，针对附件 5 的多个数据，求得切换方案最少分别需要 **16，122，178，188** 秒。

关键词:　整数规划，0-1 规划，最大值最小化

一　问题重述

1.1　问题研究背景

随着我国生产行业的快速发展，包装印刷需求量不断增加，相较于传统意义上的印刷工艺而言，柔性版印刷工艺无论从印版制作生产还是经济环保方面来说都具有突出的优势。作为印刷行业的一大发展趋势，柔板印刷效率的提升极具研究意义。使用印刷机完成柔性版印刷，需要印刷机滚筒、墨斗辊等部件的协调配合，利用印刷机插槽内的颜料印刷包装。由于只要卡槽上包含印刷某包装所需的所有墨盒，即可完成包装的印刷。由于不同的包装需要不同的颜料，在印刷完某个包装之后需要对某些插槽进行清洗且不同颜料替换的清洗时间也不完全相同。而一台清洗机一次只能清洗一台印刷设备，所以在印刷过程中减少墨盒的替换次数可以大幅度提升印刷效率[1]。

1.2　待解决问题

基于上述背景，将插槽内的墨盒取出并放置另一个墨盒的过程定义为一次切换，本文需要完成以下不同情况下地墨盒替换问题：

（1）假设只有一台清洗机，在包装印刷顺序固定而每种包装所需的墨盒顺序不影响印刷的条件下，寻找墨盒总切换次数最小的替换方案，根据附件 1 确定其切换总次数。

（2）假设只有一台清洗机，在包装印刷顺序固定，每种包装所需的墨盒顺序不影响印刷，而每种墨盒切换时间不完全相同的条件下，寻找墨盒切换时间最小的替换方案，根据附件 2 确定其切换总时间。

（3）假设只有一台清洗机，在包装印刷顺序固定，且每种包装对应的墨盒顺序也固定不变的条件下，寻找墨盒切换时间最小的替换方案，针对附件 3 给出的数据，计算最小切换时间。

（4）假设只有一台清洗机，针对附件 4 给出的数据，在包装印刷顺序不固定，而每种包装对应的墨盒顺序固定不变的条件下，研究墨盒切换时间最小的替换方案，计算墨盒的最小切换时间。

（5）与第四个问题不同的是，第五个问题将清洗机器设置为 2，计算附件 5 给出的数据中，墨盒切换时间的最小值。根据上述问题抽象出墨盒替换过程的通用处理流程如图 1：

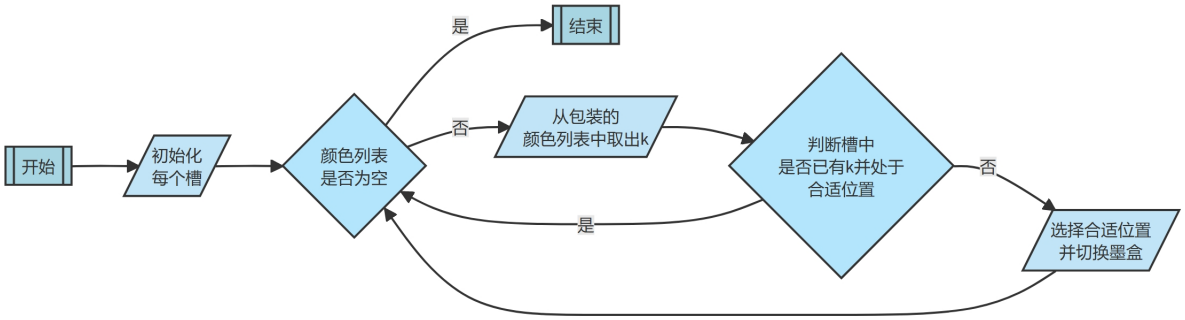

图 1: 墨盒切换通用流程

二　模型假设

1. 在印刷过程中，假设同一种颜色的墨盒有且仅有一个；

2. 假设将墨盒的切换和清洗看作一个过程；

3. 假设取出一个墨盒后必须放入另一个墨盒；

4. 假设一个插槽只能放置一个墨盒；

三　符号说明

符号	说明
$x_{i,j,k}$	包装 j 是否将墨盒 k 插入到了插槽 i
$y_{i,j,k}$	插槽 i 是否在第 j 个包装将墨盒 k 取出
N_{page}	插槽的数量
N_{pac}	包装的数量
N_{box}	墨盒的数量
pac_j	第 j 个包装需要的墨盒列表
N_{pac_j}	第 j 个包装需要的墨盒列表的墨盒数量
$pac_{j,p}$	第 j 个包装的墨盒列表里的第 p 个墨盒
N_{change}	墨盒的总切换次数
T_{change}	墨盒的总切换时间
$MATRIXchage_{k_1,k_2}$	墨盒 k_1 向 k_2 的转换时间
$z_{p,q}$	第 p 轮是否印刷第 q 个包装
$w_{i,j,k}$	第 i 轮印刷是否在第 j 个槽放置了第 k 个颜色
N_{print}	印刷的次数
$G_{i,j,k}$	第 i 次印刷时插槽 j 是否被分到了 k 组

表 1: 表符号说明表

四　问题一模型的建立与求解

4.1　问题分析

问题一要求在只有一台清洗机且固定包装印刷顺序的条件下，通过更改墨盒的替换策略，寻求墨盒替换次数的最小值。本文将上述墨盒插入、取出过程理解为序列已知的页面置换过程，进一步根据题意提炼出以下模型：对于每一个包装所需要的颜色，都必须在插槽上给其选择一个合适的位置。由于颜料的顺序不影响印刷，所以只需要保证颜料位于槽内即可，因此在替换卡槽的时候应该尽可能的优先把后续使用次数少的颜色替换掉，以保证能够取得最小的替换次数。

4.2　问题一模型的建立

针对问题一的建模，对于某个包装所需要的颜料，应先确定其是否存在于插槽上，其次决定是否需要选择一个插槽进行替换。而在问题描述中无须考虑印刷时间相关信息，仅在替换墨盒过程中需要进行清洗操作。在印刷所有的包装情况下，来求解最小替换次数，本文将某一个包装印刷过程抽象为确定其颜料序列如何在插槽上拿走和放置的过程，这里可以简化为累计取出墨盒的次数。

对于墨盒的取出操作需要用表达式来描述这个过程，所以需要对每一轮印刷过程的每个插槽进行记录，记录其存放的颜料；当颜料被替换时，上下轮的相同插槽颜色不一致。

综合考虑选择使用 0-1 规划的方式求解，本文首先引入一系列变量

$$\begin{cases} x_{i,j,k} & \text{包装 } j \text{ 是否将墨盒 } k \text{ 插入到了插槽 } i \\ y_{i,j,k} & \text{插槽 } i \text{ 是否在第 } j \text{ 个包装将墨盒 } k \text{ 取出} \end{cases} \tag{4-1}$$

上述变量中的 x_{ijk} 和 y_{ijk} 是定义待求解的变量，如下式所述：

$$x_{i,j,k} = \begin{cases} 0 & \text{，当印刷第 } j \text{ 个包装时并未将颜料 } k \text{ 放置在插槽 } i \text{ 中} \\ 1 & \text{，当印刷第 } j \text{ 个包装时将颜料 } k \text{ 放置到了插槽 } i \text{ 中} \end{cases} \tag{4-2}$$

$$y_{i,j,k}=\begin{cases}0 & \text{，当印刷第 } j \text{ 个包装时不需要将插槽 } i \text{ 中的颜料 } k \text{ 取出}\\ 1 & \text{，当印刷第 } j \text{ 个包装时需要将插槽 } i \text{ 中的颜料 } k \text{ 取出}\end{cases} \tag{4-3}$$

首先本文定义 $x_{i,j,k}$ 来讲包装、插槽和颜色串联起来，作为问题一的决策变量。其中 $y_{i,j,k}$ 依赖于 $x_{i,j,k}$ 和 $x_{i,j-1,k}$ 的取值，若 $x_{i,j-1,k}=1$ 且 $x_{i,j,k}=1$，表明对于第 j-1 个和 j 个包装，都需要用到颜料 k，而在处理第 $j-1$ 个包装时已经保证了颜料 k 一定存在于插槽内，否则不能完成印刷。同时，因为问题一的条件明确告知对于某个包装的颜料序列不影响印刷效果，则在处理第 j 个包装时，就只需要关心颜料 k 是否存在于插槽内，也就意味着不需要进行替换，此时 $y_{i,j,k}=0$；若 $x_{i,j-1,k}=0$ 且 $x_{i,j,k}=1$ 表明对于第 j-1 个包装不需要颜料 k，而第 j 个包装需要颜料 k，即意味着需要将颜料 k 插入到插槽内。但由于本文的限定，将取出前一个颜料以及插入后一个颜料当作一个单位过程，则对于替换次数只需在将包装取出时进行累计，也就是此时 $y_{i,j,k}=0$；若 $x_{i,j-1,k}=0$ 且 $x_{i,j,k}=0$ 表明对于第 j-1 个包装不需要颜料 k，而第 j 个包装也不需要颜料 k，则不需要进行替换，$y_{i,j,k}=0$；若 $x_{i,j-1,k}=1$ 且 $x_{i,j,k}=0$，表明对于第 j-1 个包装需要颜料 k，而第 j 个包装不需要颜料 k，则可能需要对其进行替换，是否替换取决于此时插槽是否有剩余空间以及替换该颜料是否为最优解，此时 $y_{i,j,k}=1$。如果下一个包装不需要颜料 k 我们就将其取出，存在着取出但不放入的风险。这违背了我们前文中定义的取出和放入当作一个整体的设定，则需要添加约束来限制这一过程，将在下文中进行介绍。

最后对所有的 y_{ijk} 累计求和就可以得到整个印刷过程中的总替换次数，如下公式所示：

$$\min\{\sum_{i=1}^{N_{page}}\sum_{j=1}^{N_{pac}}\sum_{k=1}^{N_{box}}y_{i,j,k}\} \tag{4-4}$$

当 $x_{i,j,k}$ 取 1 时，代表处理第 j 个包装时墨盒 k 已经被放置到了插槽 i 上，则可以通过遍历每一个插槽，来判断包装内部所需要的颜料是否已经存在于印刷机上，可用于限定所有包装所需要颜色一定存在于某个插槽内，上述可总结为如下公式：

$$\sum_{i=1}^{N_{page}}x_{i,j,pac_{j,q}}=1,\quad (j=1,2,...,N_{pac};q=1,2,...,N_{pac_j}) \tag{4-5}$$

而与下述公式代表了对于印刷第 j 个包装时在插槽 i 上必须有一个颜料。

$$\sum_{k=1}^{N_{box}} x_{i,j,k} = 1, \quad (i = 1, 2, ..., N_{page}; j = 1, 2, ..., N_{pac}) \tag{4-6}$$

问题的关键在于如何将上一轮结果约束到下一轮从而进行结果传递，并且识别到颜色变化。即从 $j-1$ 包装到 j 包装时该插槽颜色从没有变化为有，能否让 $y_{i,j,k}$ 标定为 1，而在其他三种情况不变。对于 $x_{i,j,k} - x_{i,j-1,k}$ 得到结果仅在上述情况插槽颜色从没用变化为有中为 1 而在其他情况下都是小于等于 0，那么可以让 $y_{i,j,k}$ 大于等于该公式，其又是 $0-1$ 变量，就会让大于等于 1 的时候，等于 1。而在其他大于等于其他小于等于 0 的结果的情况，最小化优化过程中则会尽可能的往小数字收敛，那么就会往 0 上收敛，从而使得 $y_{i,j,k}$ 不记录其他情况。

根据上述决策变量的定义，建立如下模型进行问题的求解：

$$\min\{N_{change}\}$$

$$\text{subject.to.} \quad \begin{cases} \sum_{i=1}^{N_{page}} x_{i,j,pac_{j,q}} = 1, \quad (j = 1, 2, ..., N_{pac}; q = 1, 2, ..., N_{pac_j}) & (1) \\ \sum_{k=1}^{N_{box}} x_{i,j,k} = 1, \quad (i = 1, 2, ..., N_{page}; j = 1, 2, ..., N_{pac}) & (2) \\ y_{i,j,k} = x_{i,j-1,k} * (1 - x_{i,j,k}) & \\ (i = 1, 2, ..., N_{page}; j = 1, 2, ..., N_{pac}; k = 1, 2, ..., N_{box}) & (3) \\ N_{change} = \sum_{i=1}^{N_{page}} \sum_{j=1}^{N_{pac}} \sum_{k=1}^{N_{box}} y_{i,j,k} & (4) \end{cases}$$

目标函数为 $\sum_{i=1}^{N_{page}} \sum_{j=1}^{N_{pac}} \sum_{k=1}^{N_{box}} y_{i,j,k}$，遍历并判断所有包装以及所有颜料是否在所有的插槽上进行取出操作，并统计取出次数。其中约束变量说明：

(1) 对于第 j 个包装，遍历其颜色列表，判断每个颜色是否出现在了第 i 个插槽上，其结果累加必须为 1，保证每个需要的墨盒必须且只出现在了一个插槽上。

(2) 对于第 j 个包装，以及第 i 个插槽，判断每个颜色是否出现在了该插槽上并将结果累计，保证了一个插槽只能且一定放置了一个墨盒。初始化时，求解器会根据数据和约束

条件为每一个卡槽分配一个最合理的颜色，这一点我们的约束条件都能保证。并且由于限制了每个卡槽一定有一个颜色在其上，能够保证取出墨盒和放入墨盒这一过程的整体性。

(3) 采用这样的一种约束方式，来保证只有 $x_{i,j-1,k}=1$ 且 $x_{i,j,k}=0$ 的时候 $y_{i,j,k}$ 的值为 1，其余情况下 $y_{i,j,k}$ 的值都为 0。即只有当从卡槽中取出某个颜色时替换次数加 1。

(4) 累加所有的 $y_{i,j,k}$，即为替换的总次数，也就是需要最小化的目标函数。

4.3 求解模型

4.3 算法设计

由上述分析可知算法设计如下：

Step1: 数据预处理，将附件中的数据读出并存储为模型需要的格式。

Step2: 将数据传送给 copt 求解器，添加约束, 并将最小化替换次数作为目标函数进行求解。

Step3: 分析求解器给出的结果，并分析序列的置换。

4.3 模型的求解

将已经处理的数据利用 python+copt 建立相应的线性规划模型[2]，并设置相应的参数进行求解。可以将求解的过程理解为一个暴力搜索的过程，而添加的约束作为一个判断依据，不符合约束的情况全部舍弃。最终求得附件一给出的多个数据的结果如下表所示：

附件 1	附件 1.1	附件 1.2	附件 1.3	附件 1.4	附件 1.5
切换总次数	**5**	**8**	**48**	**58**	**130**

表 2: 附件 1 切换次数统计

下图给出了附件 1.2 的数据墨盒序列的变更情况，作为例子来展示问题一墨盒替换过程。

图 2: 附件 1.2 墨盒变更示意图

五　问题二模型的建立与求解

5.1　问题分析

问题二要求在只有一台清洗机，固定包装种类印刷顺序以及每种包装对应的墨盒顺序也固定的条件下，更改墨盒的替换策略以求得墨盒替换时间的最小值。与问题一不同的是，最终得到的目标变量为墨盒替换总时间，而不是单纯的替换总次数。由于颜料的顺序不影响印刷，所以只需要保证颜料位于槽内即可，因此在满足必要的约束条件下，替换卡槽的时候应该尽可能的优先把墨盒替换时间小的墨盒替换掉，以保证能够取得最小的墨盒替换时间。

5.2　模型的建立

在本文提出的方法中，同 4.1 中对于墨盒替换策略的理解，将墨盒替换的过程提炼出以下模型：用变量 y_{ijk} 表示第 j 个包装到第 $j+1$ 个包装是否进行墨盒替换。对于第 $j+1$

个包装所需要的墨盒 k_1 和第 j 个包装所需要的墨盒 k_2，若 k_1 和 k_2 相等，则不需要进行替换，变量 y 的值为 0。若 k_1 和 k_2 不相等，则表示变量 y 的值为 1，表示第 j 个包装到第 $j+1$ 个包装需要进行墨盒替换，墨盒切换需要花费的时间用 $MATRIXchange_{k_2,k_1}$ 来表示在插槽上由墨盒 k_2 替换到 k_1 所需要的切换时间。在满足所有的包装种类打印顺序以及所需的墨盒要求的条件下，计算出总共需要花费的时间为 T_{change}, 在求解过程中出现的 T_{change} 的最小值即为该问题的最优解。对于问题二的模型建立，应先判断对于同一个插槽，在打印前后两个包装种类时该插槽上的墨盒是否需要替换。与问题一不同之处在于问题二需要考虑切换墨盒所需的时间，在满足所有包装种类的印刷顺序的基础上求出最小的墨盒切换所需的时间之和。其中，$MATRIXchange_{k_2,k_1}$ 由附件 2 给出，代表从颜料 k_1 到颜料 k_2 的切换过程需要的时间。

根据上述决策变量的定义，建立模型如下：

$$\min\{T_{change}\}$$

$$\text{subject.to.} \quad \begin{cases} \sum_{i=1}^{N_{page}} x_{i,j,pac_{j,q}} = 1, \quad (j = 1,2,...,N_{pac}; q = 1,2,...,N_{pac_j}) & (1) \\ \sum_{k=1}^{N_{box}} x_{i,j,k} = 1, \quad (i = 1,2,...,N_{page}; j = 1,2,...,N_{pac}) & (2) \\ T_{change} = \sum_{i=1}^{N_{page}} \sum_{j=1}^{N_{pac}} \sum_{k_1}^{N_{box}} \sum_{k_2}^{N_{box}} x_{i,j,k_1} * x_{i,j+1,k_2} * MATRIXchange_{k_1,k_2}) & (3) \end{cases}$$

对于第 i 个插槽来说，第 j 个包装产品和第 $j+1$ 个包装的颜料 k_1, k_2，当 x_{ijk_1} 和 $x_{i,j+1,k_2}$ 的值都为 1 的时候, 表示在第 j 个产品的第 i 个插槽上的墨盒为 k_1, 同时在第 $j+1$ 个产品的第 i 个插槽上的墨盒为 k_2。在 x_{ijk_1} 和 $x_{i,j+1,k_2}$ 的值都为 1 的基础上判断 x_{ijk_1} 和 $x_{i,j+1,k_2}$ 中的 k_1 和 k_2 的值是否相同,如果 k_1 不等于 k_2,表示判断第 i 个插槽需要替换,那么 $x_{i,j,k_1} * x_{i,j-1,k_2} * MATRIXchange_{k_2,k_1}$ 的值为 $MATRIXchange_{k_1,k_2}$; 如果 k_1 等于 k_2, 表示判断第 i 个插槽不需要替换, 那么 $x_{i,j,k_1} * x_{i,j+1,k_2} * MATRIXchange_{k_1,k_2}$ 的值为 0。综合以上分析在发生单步替换时所需的时间就可以表示为 $x_{i,j,k_1} * x_{i,j-1,k_2} * MATRIXchange_{k_1,k_2}$。目标函数为 $\sum_{i=1}^{N_{page}} \sum_{j=1}^{N_{pac}} \sum_{k_1}^{N_{box}} \sum_{k_2}^{N_{box}} x_{i,j,k_1} * x_{i,j+1,k_2} * MATRIXchange_{k_1,k_2}$，即将所有包装的所有墨盒在插槽上进行替换操作所需时间的累计。其中约束变量说明：

(1)(2) 与问题一中的约束变量说明相同，表示特定的第 j 个包装的插槽 i 的所有 k 墨盒颜色中至多有一种颜色。与问题一不同的是，约束 (3) 在插槽发生墨盒替换时加入了 $MATRIXchange_{k_1,k_2}$ 的时间变量。

5.3 求解模型

5.3 算法设计

由上述分析可知算法设计如下：

Step1: 数据预处理，将附件中的数据读出并存储为模型需要的格式。

Step2: 将数据传送给 copt 求解器，添加约束, 并将最小化替换时间作为目标函数进行求解。

Step3: 分析求解器给出的结果，并分析序列的置换。

5.3 模型的求解

将已经处理的数据利用 python+copt 建立相应的线性规划模型，并设置相应的参数进行求解。可以将求解的过程理解为一个暴力搜索的过程，而添加的约束作为一个判断依据，不符合约束的情况全部舍弃。最终求得附件二给出的多个数据的结果如下表所示：

附件 2	附件 2.1	附件 2.2	附件 2.3	附件 2.4	附件 2.5
切换总时间	**32**	**51**	**111**	**203**	**326**

表 3: 附件 2 切换时间统计

图 3 给出了附件 2.2 的数据墨盒序列的变更情况，作为例子来展示问题二墨盒替换过程。

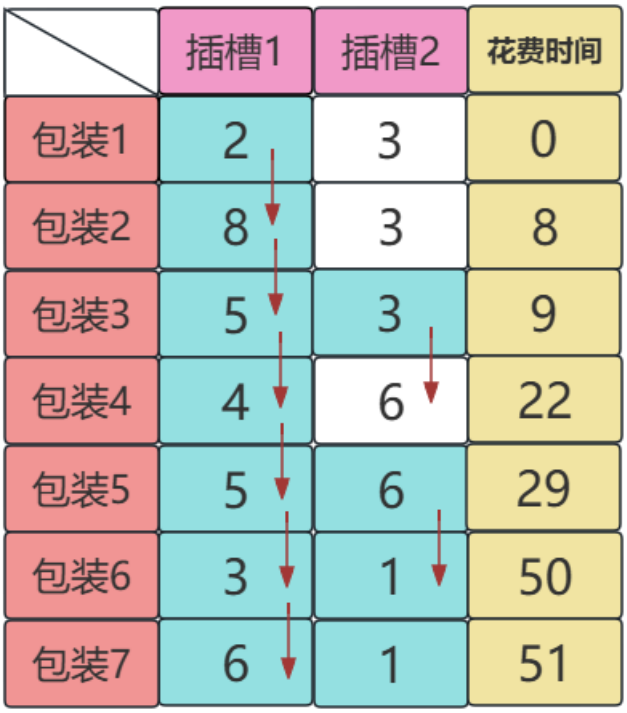

图 3: 附件 2.2 墨盒变更示意图

六　问题三模型的建立与求解

6.1　问题分析

问题三要求在只有一台清洗机，固定包装印刷顺序以及每种包装对应的墨盒顺序也固定的条件下，通过制定替换策略来切换墨盒以求得墨盒替换时间的最小值。与问题一和二不同的是，问题三需要保证印刷机上的墨盒顺序必须与包装的颜色顺序一致。在满足问题二约束的基础上，必须添加新的约束来满足包装内墨盒的印刷顺序和印刷机上墨盒相对顺序的一致性。计算出整体替换策略总共的墨盒切换时间 T_{change}，求解过程中出现的 T_{change} 的最小值即为该问题的最优解。

6.2　问题三模型的建立

在本文提出的方法中，基于问题一和问题二对于墨盒替换策略的理解，将问题三的墨盒替换过程提炼出以下模型：用变量 y_{ijk} 表示第 j 个包装到第 $j+1$ 个包装是否进行墨盒

替换。印刷机从墨盒 k_2 切换到 k_1 需要花费的时间用 $MATRIXchange_{k_2,k_1}$ 来表示。在此基础上新增包装种类 j 所需的墨盒颜色顺序 pac_j，用 $pac_{j,p}$ 来表示该包装列表中第 p 个所需的墨盒颜色。检查该颜色墨盒所在的插槽位置 i_1, 对于该包装装墨盒颜色序列的下一个墨盒颜色 $pac_{j,p+1}$，检查该颜色墨盒所在的插槽位置 i_2，必须满足 i_2 的值大于 i_1。令该包装所需的墨盒满足上述条件即可固定其在印刷机上顺序。在满足所有的包装种类打印顺序以及包装种类内部墨盒颜色顺序固定的条件下，计算出总共需要花费的时间为 T_{change}, 在求解过程中出现的 T_{change} 的最小值即为该问题的最优解。

根据上述决策变量，模型的建立如下：

$$\min\{T_{change}\}$$

$$\text{subject.to.} \quad \begin{cases} \sum_{i=1}^{N_{page}} x_{i,j,pac_{j,q}} = 1, \quad (j = 1,2,...,N_{pac}; q = 1,2,...,N_{pac_j}) & (1) \\ \sum_{k=1}^{N_{box}} x_{i,j,k} = 1, \quad (i = 1,2,...,N_{page}; j = 1,2,...,N_{pac}) & (2) \\ \sum_{q=1}^{N_{pac_j}-1} \sum_{i=1}^{N_{page}} x_{i,j,pac_{j,q}} * i < \sum_{q=1}^{N_{pac_j}-1} \sum_{i=1}^{N_{page}} x_{i,j,pac_{j,q+1}} * i, \quad (j = 1,2,...,N_{pac}) & (3) \\ T_{change} = \sum_{i=1}^{N_{page}} \sum_{j=1}^{N_{pac}} \sum_{k_1}^{N_{box}} \sum_{k_2}^{N_{box}} x_{i,j,k_1} * x_{i,j+1,k_2} * MATRIXchange_{k_1,k_2}) & (4) \end{cases}$$

针对第 j 个包装产品中的墨盒顺序，如果第 q 个墨盒所在的插槽序号为 i，$x_{i,j,pac_{j,q}}$ 的值为 1，那么 $x_{i,j,pac_{j,q}} * i$ 的值为 i(即第 q 个墨盒所在的插槽序号为 i)；如果第 q 个墨盒所在的插槽序号不为 i，$x_{i,j,pac_{j,q}}$ 的值为 0，那么 $x_{i,j,pac_{j,q}} * i$ 的值为 0。因此 $\sum_{i=1}^{N_{page}} x_{i,j,pac_{j,q}} * i$ 所表示的值必定是第 j 个包装的第 q 个墨盒所在的插槽序号 i_1。因此也可以计算出第 q 个墨盒所在的插槽序号 i_2，由计算得到的两个插槽序号来添加约束 (3)。目标函数为 $\sum_{i=1}^{N_{page}} \sum_{j=1}^{N_{pac}} \sum_{k_1}^{N_{box}} \sum_{k_2}^{N_{box}} x_{i,j,k_1} * x_{i,j+1,k_2} * MATRIXchange_{k_1,k_2}$，即将所有包装的所有墨盒在插槽上进行替换操作所需时间的累计。其中约束变量 (4) 说明：

(1)(2)(4) 与问题二中的约束变量说明相同。与问题二不同的是，约束 (3) 表示第 j 个包装的第 q 个墨盒颜色 $pac_{j,q}$ 所在的插槽序号 i_1 必须小于第 j 个包装的第 $q+1$ 个墨盒颜色 $pac_{j,q+1}$ 所在的插槽序号 i_2，即以此来满足印刷机上被印刷颜色顺序和包装颜色列表

的相对一致性。

6.3 求解模型

6.3 算法设计

由上述分析可知算法设计如下：

Step1: 数据预处理，将附件中的数据读出并存储为模型需要的格式。

Step2: 将数据传送给 copt 求解器，添加约束, 并将最小化替换时间作为目标函数进行求解。

Step3: 分析求解器给出的结果，并分析序列的置换。

6.3 模型的求解

将已经处理的数据利用 python+copt 建立相应的线性规划模型，并设置相应的参数进行求解。可以将求解的过程理解为一个暴力搜索的过程，而添加的约束作为一个判断依据，不符合约束的情况全部舍弃。最终求得附件三给出的多个数据的结果如下表所示：

附件 3	附件 3.1	附件 3.2	附件 3.3	附件 3.4
切换总时间	**56**	**184**	**245**	**288**

表 4: 附件 3 切换时间统计

图 4 给出了附件 3.1 的数据墨盒序列的变更情况，作为例子来展示问题三墨盒替换过程。

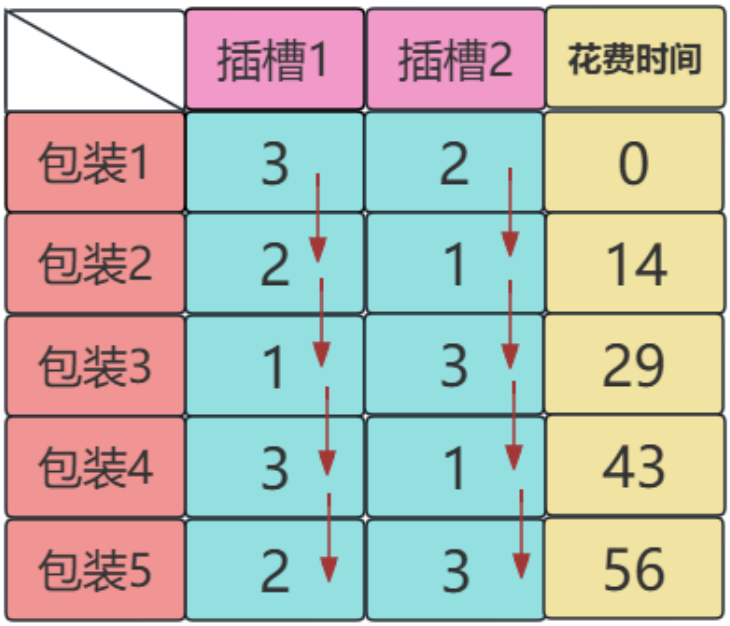

图 4: 附件 3.1 墨盒变更示意图

七　问题四模型的建立与求解

7.1　问题分析

问题四要求在只有一台清洗机，且每种包装对应的墨盒顺序固定的条件下，通过制定合理的墨盒切换顺序，以求得墨盒替换整体时间的最小值。与问题三不同的是，问题四允许改变每种包装的打印顺序。在满足所有约束下计算出所有墨盒切换的总体时间 T_{change}，在求解过程中出现的 T_{change} 的最小值即为该问题的最优解。

7.2　问题四模型的建立

本文在针对问题四的求解时引入下列新的变量：

$$\begin{cases} w_{i,j,k} & \text{第 } i \text{ 次印刷是否将墨盒 } k \text{ 插入到了插槽 } j \\ z_{p,q} & \text{第 } p \text{ 轮印刷是否印刷包装 } q \end{cases} \tag{7-1}$$

根据定义 $w_{i,j,k}$ 的取值如下：

$$w_{i,j,k} = \begin{cases} 0 & ，当第\ i\ 轮印刷时并未将\ k\ 放置在插槽\ j\ 中 \\ 1 & ，当第\ i\ 轮印刷时将颜料\ k\ 放置到了插槽\ j\ 中 \end{cases} \tag{7-1}$$

$$z_{p,q} = \begin{cases} 0 & ，第\ p\ 轮印刷并未印刷包装\ q \\ 1 & ，第\ p\ 轮印刷印刷包装\ q \end{cases} \tag{7-2}$$

在本文提出的方法中，基于问题一、问题二、问题三对于墨盒替换策略的理解，将问题四的墨盒替换过程提炼出以下模型：用变量 $z_{p,q}$ 来表示第 p 轮印刷的是 q 包装，用变量 $w_{i,j,k_1} * w_{i+1,j,k_2}$ 表示第 i 轮包装到第 $i+1$ 轮包装是否进行墨盒替换，在 w_{i,j,k_1} 和 w_{i+1,j,k_2} 的值同为 1 时表示在第 i 轮印刷中插槽 j 进行了墨盒替换。墨盒 k_1 切换到 k_2 需要花费的时间用 $MATRIXchange_{k_1,k_2}$ 来表示。变量 $pac_{q,id}$ 表示第 q 个包装的第 id 个位置的墨盒颜色，检查该颜色墨盒所在的插槽位置 j_1，对于该包装墨盒颜色序列的下一个墨盒颜色 $pac_{q,id+1}$，检查该颜色墨盒所在的插槽位置 j_2，必须满足 j_2 的值大于 j_1。在满足同一包装种类内部墨盒颜色顺序的条件下得到一种墨盒替换的策略，计算出墨盒切换需要花费的总时间为 T_{change}，在求解过程中出现的 T_{change} 的最小值即为该问题的最优解。

根据上述决策变量的定义模型的建立如下：

$$\min\{T_{change}\}$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{N_{page}} w_{i,j,k} <= 1, \quad (i = 1,2,...,N_{print}; k = 1,2,...,N_{box}) & (1) \\ \sum_{k=1}^{N_{box}} w_{i,j,k} = 1, \quad (i = 1,2,...,N_{print}; j = 1,2,...,N_{page}) & (2) \\ \sum_{p=1}^{N_{print}} z_{p,q} = 1, \quad (q = 1,2,...,N_{print}) & (3) \\ \sum_{q=1}^{N_{pac}} z_{p,q} = 1, \quad (p = 1,2,...,N_{pac}) & (4) \\ z_{p,q} - \sum_{j=1}^{N_{page}} w_{p,j,k} <= 0, \quad (p = 1,2,...,N_{print}; k \in pac_q) & (5) \\ z_{p,q}(\sum_{j=1}^{N_{page}} w_{p,j,pac_{q,id}} * j - \sum_{j=1}^{N_{page}} w_{p,j,pac_{q,id+1}} * j) <= 0, \\ (p = 1,2,...,N_{pac}; q = 1,2,...,N_{pac}, id = 1,2,...,N_{pac_q}) & (6) \\ T_{change} = \sum_{p=1}^{N_{pac}} z_{p,q_1} * z_{p+1,q_2} * w_{p,j,k_1} * w_{p+1,j,k_2} * MATRIXchange_{k_2,k_1}) \\ (k_1 = 1,2,...,N_{pac_{q_1}}; k_2 = 1,2,...,N_{pac_{q_2}}; q_1 = 1,2,...,N_{pac}; q_2 = 1,2,...,N_{pac}) & (7) \end{cases}$$

约束条件（1）限定了每次印刷时每个颜色至多被放置在一个插槽内。

约束条件（2）限定了每次印刷时每个槽一定会放置一个颜料。与前三个问题一样，把取出与插入当作一个统一的过程，且在第一轮印刷时就根据所有包装的印刷顺序将插槽装满。

约束（3）限定了每个包装一定会被印刷一次。

约束（4）限定了每轮一定会印刷一个包装。

约束（5）保证了若第 p 轮印刷第 q 个包装，那么 q 包装列表内的每一个颜料都必须在插槽内。如该式所属，减号前后两式中前者表示第 p 轮是否印刷第 q 个包装，后者表示，遍历每一个插槽并判断是否有颜色 k, 并累加其结果。由于约束条件（1）（2）限定了一个槽内只能存在一个颜色，并且一个颜色至多在一个槽内存在，因此对于确定的轮次后者的值一定为 0 或 1；则该约束表示前者为 1 时，后者一定不能为 0，即实现了印刷第 q 个包装

时其所需的颜色一定在槽内。

约束（6）是在问题三的第三个约束的基础上做出了改动，保证第 p 轮印刷时，颜料的排列一定与照题目给定的顺序一致。

约束（7）定义了本模型求解的目标函数，即统计每轮印刷时每个插槽的切换时间（不切换时 $MATRIXchange_{k2,k1} = 0$）。

7.3 求解模型

7.3 算法设计

由上述分析可知算法设计如下：

Step1: 数据预处理，将附件中的数据读出并存储为模型需要的格式。

Step2: 将数据传送给 copt 求解器，添加约束, 并将最小化替换时间作为目标函数进行求解。

Step3: 分析求解器给出的结果，并分析序列的置换。

7.3 模型的求解

将已经处理的数据利用 python+copt 建立相应的线性规划模型，并设置相应的参数进行求解。可以将求解的过程理解为一个暴力搜索的过程，而添加的约束作为一个判断依据，不符合约束的情况全部舍弃。最终求得附件 4 给出的多个数据的结果如下表所示：

附件 4	附件 4.1	附件 4.2	附件 4.3	附件 4.4
切换总时间	**20**	**37**	**65**	**196**

表 5: 附件 2 切换时间统计

图 5 给出了附件 4.1 的数据墨盒序列的变更情况，作为例子来展示问题四墨盒替换过程。

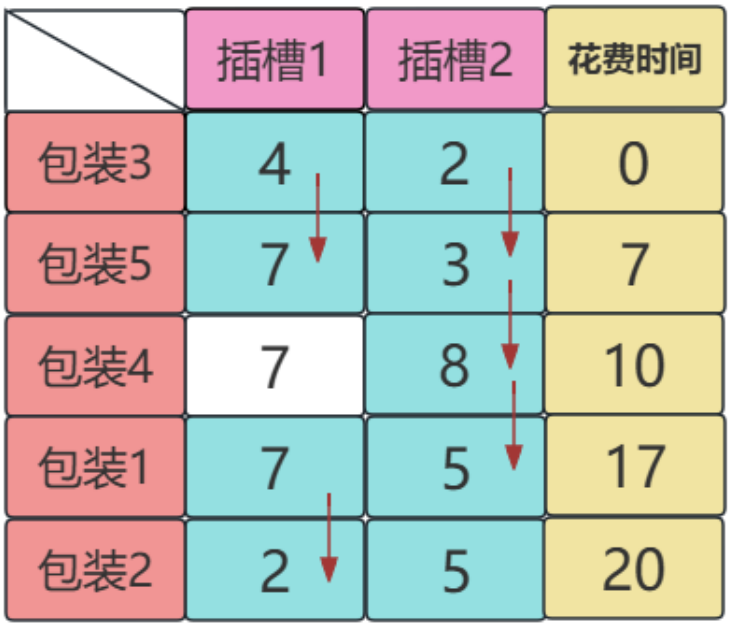

图 5: 附件 4.1 墨盒变更示意图

八　问题五模型的建立与求解

8.1　问题分析

问题五要求在有两台清洗机，只固定每种包装对应的墨盒顺序，通过更改墨盒的替换策略，以求得墨盒替换时间的最小值。与问题四不同的是，问题四的清洗机数量从一台增加到两台，意味着在每一轮印刷过程中可以同时清洗不同的插槽位置，以达到节省切换时间的目的。在满足所有约束下计算出整体墨盒切换策略总共需要花费的时间为 T_{change}，在求解过程中出现的 T_{change} 的最小值即为该问题的最优解。

8.2 问题五模型的建立

本文在针对问题五的求解时引入下列新的变量：

$$\begin{cases} G_{i,j,k} & \text{第 } i \text{ 次印刷时插槽 } j \text{ 是否被分到了 } k \text{ 组} \\ C_{p,q} & \text{第 } i \text{ 次印刷第 } j \text{ 组的清洗时间} \\ T_i & \text{第 } i \text{ 次印刷所用的清洗时间} \end{cases} \tag{8-1}$$

根据定义 $G_{i,j,k}$ 的取值如下：

$$G_{i,j,k} = \begin{cases} 0 & \text{，第 } i \text{ 次印刷时插槽 } j \text{ 未被分到了 } k \text{ 组} \\ 1 & \text{，第 } i \text{ 次印刷时插槽 } j \text{ 被分到了 } k \text{ 组} \end{cases} \tag{8-2}$$

问题五相对于问题四，清洗机的数量增加到两台，这就导致在每轮印刷结束之后清洗时间不应该直接相加。由于有两台清洗机，则应该把每轮所有的槽分为两组，并且尽量地让两组的清洗时间和相等。那么这一轮的清洗时间就应该由两个切换分组所需时间的最大值决定。于是问题就转化为如何将需要清洗的插槽分为切换时间和尽量相等的两组。在本文提出的方法中，基于问题一、问题二、问题三、问题四对于墨盒替换策略的理解，将问题五的墨盒替换过程提炼出以下模型：用变量 $w_{i,j,k_1} * w_{i+1,j,k_2}$ 表示第 i 轮包装到第 $i+1$ 轮包装是否进行墨盒替换, 只有 w_{i,j,k_1} 和 w_{i+1,j,k_2} 的值都为 1 时，墨盒 k_1 切换到 k_2 需要花费的时间用 $MATRIXchange_{k_1,k_2}$ 来表示。变量 $G_{i,j,k}$ 表示第 i 次印刷时插槽 j 是否被分到了 k 组。计算在一轮印刷中每个插槽上发生的墨盒替换的时间得到每一轮墨盒替换所需的总时间，再通过累计每一轮墨盒替换所需的总时间得到整个替换策略中发生墨盒替换所需的总时间 T_{change}，在求解过程中出现的 T_{change} 的最小值即为该问题的最优解。

根据上述决策变量的定义模型的建立如下：

$$T_{change} = \min\{ \sum_{i=1}^{N_{print}} T_i \}$$

subject.to.

$$\begin{cases}
\sum_{j=1}^{N_{page}} w_{i,j,k} <= 1, \quad (i = 1,2,...,N_{print}; k = 1,2,...,N_{box}) & (1) \\
\sum_{k=1}^{N_{box}} w_{i,j,k} = 1, \quad (i = 1,2,...,N_{print}; j = 1,2,...,N_{page}) & (2) \\
\sum_{p=1}^{N_{print}} z_{p,q} = 1, \quad (q = 1,2,...,N_{print}) & (3) \\
\sum_{q=1}^{N_{pac}} z_{p,q} = 1, \quad (p = 1,2,...,N_{pac}) & (4) \\
z_{p,q} - \sum_{j=1}^{N_{page}} w_{p,j,k} <= 0, \quad (p = 1,2,...,N_{print}; k \in pac_q) & (5) \\
z_{p,q}(\sum_{j=1}^{N_{page}} w_{p,j,pac_{q,id}} * j - \sum_{j=1}^{N_{page}} w_{p,j,pac_{q,id+1}} * j) <= 0, & \\
(p = 1,2,...,N_{pac}; q = 1,2,...,N_{pac}, id = 1,2,...,N_{pac_q}) & (6) \\
\sum_{k=1}^{N_{group}} G_{i,j,k} = 1, \quad (i = 1,2,...,N_{print}; j = 1,2,...,N_{page}) & (7) \\
T_i >= \sum_{j=1}^{N_{page}} G_{i,j,p} * w_{i,j,k_1} * w_{i+1,j,k_2} * MATRIXchange_{k_1,k_2}, & \\
(i = 1,2,...,N_{print} - 1; p = 1,2; k_1 = 1,2,...,N_{box}; k_2 = 1,2,...,N_{box}) & (8)
\end{cases}$$

约束条件（1）限定了每次印刷时每个颜色至多被放置在一个插槽内。

约束条件（2）限定了每次印刷时每个槽一定会放置一个颜料。与前个问题一样，把取出与插入当作一个统一的过程，且在第一轮印刷时就根据所有包装的印刷顺序将插槽装满。

约束（3）限定了每个包装一定会被印刷一次。

约束（4）限定了每轮一定会印刷一个包装。

约束（5）保证了若第 p 轮印刷第 q 个包装，那么 q 包装列表内的每一个颜料都必须在插槽内。

约束（6）保证第 p 轮印刷时，颜料的排列一定与照题目给定的顺序一致。

约束（7）保证每轮印刷结束后每个槽都被分配给 1 和 2 中的两个清洗机之一。

约束（8）定义了本模型求解的目标函数，即统计每轮印刷时每个插槽的切换时间（不切换时 $MATRIXchange_{k2,k1} = 0$）。由于每轮的清洗时间应由两组清洗机中时间最长的决定，因此采用该约束即可以代表取两组中的最大值。

8.3 求解模型

8.3 算法设计

由上述分析可知算法设计如下：

Step1: 数据预处理，将附件中的数据读出并存储为模型需要的格式。

Step2: 将数据传送给 gurobi 求解器，添加约束，并将最小化替换时间作为目标函数进行求解。

Step3: 分析求解器给出的结果，并分析序列的置换。

8.3 模型的求解

由于本文给出的模型已经将每轮所有的插槽进行分组，T_j 代表第 j 轮的清洗时间，目标是最小化 T_j 的和，则求解器会在求解过程中尽量的最小化 T_j 的值，也就是最小化两台清洗机中最大的那个，从而实现尽量地让两台清洗机的清洗时间相同[3]。将已经处理的数据利用 c++ 和 gurobi 建立相应的数学模型，并设置相应的参数进行求解。可以将求解的过程理解为一个暴力搜索的过程，而添加的约束作为一个判断依据，不符合约束的情况全部舍弃。最终求得附件 5 给出的多个数据的结果如下表所示：

附件 5	附件 5.1	附件 5.2	附件 5.3	附件 5.4
切换总时间	**16**	**122**	**178**	**188**

表 6: 附件 5 切换时间统计

图 6 给出了附件 5.1 的数据墨盒序列的变更情况，作为例子来展示问题五墨盒替换过程。

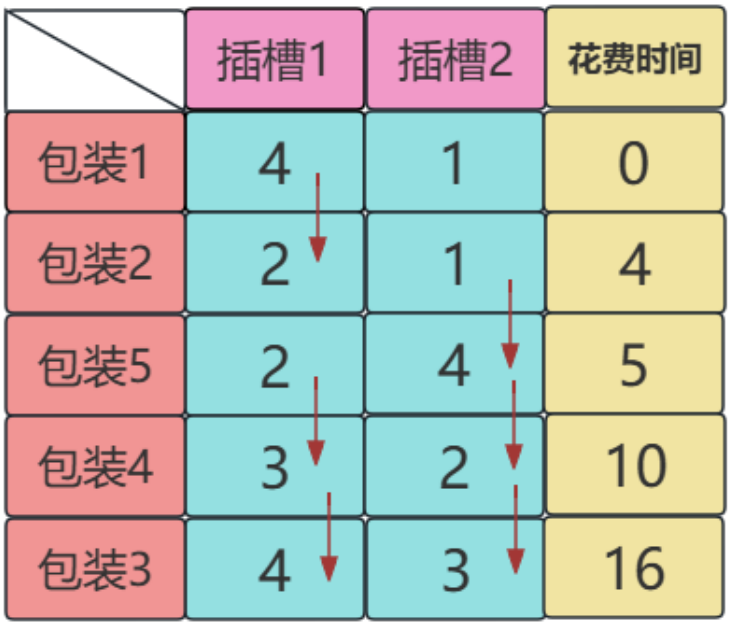

图 6: 附件 5.1 墨盒变更示意图

九　模型的评价、改进与推广

9.1　模型的优点

（1）本文针对柔性版印刷插槽内颜料的最小替换次数，建立了 0-1 整数规划模型[4]，能够有效的求解出问题中情况下最小替换次数；（2）针对题目中给出的条件，本文建立的模型能够有效地求解出最短时间，从数学的角度直观、有效地解决了问题，且易于理解。

9.2　模型的缺点

（1）本文在求解大规模的数据时存在求解时间慢的问题；（2）模型建立时引入了非线性的约束，导致使用 copt 求解时存在需要升维来解决，增加了代码的冗余度。

9.3 模型的改进

（1）在未来的研究中，尽可能的采用线性约束来建模；（2）在未来的研究中对求解算法进一步提高结果精度。

9.4 模型的推广

本文对于柔性版印刷磨合的替换策略进行了研究，对实际生活中其他的替换策略有借鉴意义。

参考文献

[1] 康启来. 柔性版印刷工艺的若干知识 [J]. 广东印刷,2005,000(005):39-41.DOI:10.3969/j.issn.1005-7463.2005.05.020.

[2] 刘兴禄，优化 | 手把手教你学会杉树求解器 (COPT) 的安装、配置与测试，https://blog.csdn.net/HsinglukLiu/article/details/123142656,2022/2/26.

[3] 司守奎, 孙玺菁. 数学建模算法与应用 [M]. 国防工业出版社,2011.

[4] 谭瑛，高慧敏，曾建潮. 求解整数规划问题的微粒群算法 [J]. 系统工程理论与实践,2004,(05):126-129.

附录

```
    %%%问题一

import pandas as pd
import numpy as np
import coptpy as copt
filePath="D:/workPlace/modelMath/data/附件1/"
# file=filePath+'Ins1_5_10_2.xlsx'
# file=filePath+'Ins2_7_10_3.xlsx'
# file=filePath+'Ins3_10_50_15.xlsx'
# file=filePath+'Ins4_20_50_10.xlsx'
file=filePath+'Ins5_30_60_10.xlsx'
# 读取xlsx文件
dfNum = pd.read_excel(file)
dfCollate = pd.read_excel(file, sheet_name=1)
dfCollate['所需墨盒编号'] = dfCollate['所需墨盒编号'].apply(lambda x: [i - 1 for i in eval(x)])
packageID=[]
inkID=[]
insertID=[]
for i in dfNum['包装种类编号'].dropna():
    packageID.append(i)
for i in dfNum['墨盒编号'].dropna():
    inkID.append(i)
for i in dfNum['插槽序号'].dropna():
    insertID.append(i)
I=len(insertID)
J=len(packageID)
K=len(inkID)
# 初始化 COPT 环境和模型
env = copt.Envr()
model = env.createModel()

# 创建决策变量 x 和 s
x = model.addVars(I, J, K, vtype=copt.COPT.BINARY, nameprefix='x')
y = model.addVars(I, J, K, vtype=copt.COPT.BINARY, nameprefix='y')

#最小化y
model.setObjective(copt.quicksum(y[i, j, k] for i in range(I) for j in range(J) for k in range(K)), copt.COPT
    .MINIMIZE)

#所有包装所需要颜色内必须要有
for j in range(J):
    for k in dfCollate.iloc[j,1]:
        model.addConstr(copt.quicksum(x[i, j, k] for i in range(I)) == 1)

```

```
#j包装时候i槽必须有一个颜色
for j in range(J):
    for i in range(I):
        model.addConstr(copt.quicksum(x[i, j, k] for k in range(K)) == 1)


#颜色判断变化
for j in range(1,J):
    for i in range(I):
#         for k in dfCollate.iloc[j,1]:
        for k in range(K):
        #当前需要之前没有，相当于放进去墨盒。
            model.addConstr(y[i,j,k] >= x[i,j,k] - x[i,j-1,k])

model.solve()
```

```
    %%%问题二

import pandas as pd
import numpy as np
import coptpy as copt
filePath="D:/workPlace/modelMath/data/附件2/"
file=filePath+'Ins1_5_10_3.xlsx'
#file=filePath+'Ins2_7_10_2.xlsx'
# file=filePath+'Ins3_10_30_10.xlsx'
# file=filePath+'Ins4_20_40_10.xlsx'
# file=filePath+'Ins5_30_60_10.xlsx'
# 读取xlsx文件
# 读取xlsx文件
dfNum = pd.read_excel(file)
dfCollate = pd.read_excel(file, sheet_name=1)
dfClear = pd.read_excel(file, sheet_name=2)
dfCollate['所需墨盒编号'] = dfCollate['所需墨盒编号'].apply(lambda x: [i - 1 for i in eval(x)])
packageID=[]
inkID=[]
insertID=[]
for i in dfNum['包装种类编号'].dropna():
    packageID.append(i)
for i in dfNum['墨盒编号'].dropna():
    inkID.append(i)
for i in dfNum['插槽序号'].dropna():
    insertID.append(i)
I=len(insertID)
J=len(packageID)
K=len(inkID)
```

```
matrixTime=dfClear.iloc[:,1:].values.tolist()
# 初始化 COPT 环境和模型
env = copt.Envr()
model = env.createModel()

# 创建决策变量 x 和 s
x = model.addVars(I, J, K, vtype=copt.COPT.BINARY, nameprefix='x')
y = model.addVars(I, J, K, vtype=copt.COPT.BINARY, nameprefix='y')
#拿与放置的变量存储
TakeSet = model.addVars(I, J, K, K, vtype=copt.COPT.BINARY, nameprefix='TakeSet')

#最小化y
model.setObjective(copt.quicksum(TakeSet[i, j, t,s]*matrixTime[t][s] for i in range(I) for j in range(1,J)
                            for t in range(K) for s in range(K)), copt.COPT.MINIMIZE)

#所有包装所需要颜色内必须要有
for j in range(J):
    for k in dfCollate.iloc[j,1]:
        model.addConstr(copt.quicksum(x[i, j, k] for i in range(I)) == 1)

#j包装时候i槽必须有一个颜色
for j in range(J):
    for i in range(I):
        model.addConstr(copt.quicksum(x[i, j, k] for k in range(K)) == 1)


#识别拿去和放置
for j in range(1,J):
    for i in range(I):
        for t in range(K):
            for s in range(K):
                model.addConstr(TakeSet[i,j,t,s]>= y[i,j,s]+x[i,j-1,t]-1)
                model.addConstr(TakeSet[i,j,t,s]<= y[i,j,s])
                model.addConstr(TakeSet[i,j,t,s]<= x[i,j-1,t])

#颜色判断变化
for j in range(1,J):
    for i in range(I):
#         for k in dfCollate.iloc[j,1]:
        for k in range(K):
        #当前需要之前没有，相当于放进去墨盒。
            model.addConstr(y[i,j,k] >= x[i,j,k] - x[i,j-1,k])

model.solve()
```

```
    %%%问题三

import pandas as pd
import numpy as np
import coptpy as copt
filePath="D:/workPlace/modelMath/data/附件3/"
file=filePath+'Ins1_5_5_2.xlsx'
# file=filePath+'Ins2_10_30_10.xlsx'
# file=filePath+'Ins3_20_50_10.xlsx'
# file=filePath+'Ins4_30_60_10.xlsx'
# 读取xlsx文件
# 读取xlsx文件
dfNum = pd.read_excel(file)
dfCollate = pd.read_excel(file, sheet_name=1)
dfClear = pd.read_excel(file, sheet_name=2)
dfCollate['所需墨盒编号'] = dfCollate['所需墨盒编号'].apply(lambda x: [i - 1 for i in eval(x)])
packageID=[]
inkID=[]
insertID=[]
for i in dfNum['包装种类编号'].dropna():
    packageID.append(i)
for i in dfNum['墨盒编号'].dropna():
    inkID.append(i)
for i in dfNum['插槽序号'].dropna():
    insertID.append(i)
I=len(insertID)
J=len(packageID)
K=len(inkID)
matrixTime=dfClear.iloc[:,1:].values.tolist()
# 初始化 COPT 环境和模型
env = copt.Envr()
model = env.createModel()

# 创建决策变量 x 和 s
x = model.addVars(I, J, K, vtype=copt.COPT.BINARY, nameprefix='x')
y = model.addVars(I, J, K, vtype=copt.COPT.BINARY, nameprefix='y')
#拿与放置的变量存储
TakeSet = model.addVars(I, J, K, K, vtype=copt.COPT.BINARY, nameprefix='TakeSet')

#最小化y
model.setObjective(copt.quicksum(TakeSet[i, j, t,s]*matrixTime[t][s] for i in range(I) for j in range(1,J)
                            for t in range(K) for s in range(K)), copt.COPT.MINIMIZE)

#所有包装所需要颜色内必须要有
for j in range(J):
```

```
    for k in dfCollate.iloc[j,1]:
        model.addConstr(copt.quicksum(x[i, j, k] for i in range(I)) == 1)

#j包装时候i槽必须有一个颜色
for j in range(J):
    for i in range(I):
        model.addConstr(copt.quicksum(x[i, j, k] for k in range(K)) == 1)


# y = model.addVars(I, J, K, vtype=copt.COPT.BINARY, nameprefix='y')
#识别拿去和放置
for j in range(1,J):
    for i in range(I):
        for t in range(K):
            for s in range(K):
                model.addConstr(TakeSet[i,j,t,s]>= y[i,j,s]+x[i,j-1,t]-1)
                model.addConstr(TakeSet[i,j,t,s]<= y[i,j,s])
                model.addConstr(TakeSet[i,j,t,s]<= x[i,j-1,t])
#设置顺序约束
for j in range(J):
    temp=dfCollate.iloc[j,1]
    for k in range(1,len(temp)):
        model.addConstr(copt.quicksum(i*x[i,j,temp[k-1]]-i*x[i,j,temp[k]] for i in range(I)) + 1 <=0)

#颜色判断变化
for j in range(1,J):
    for i in range(I):
#         for k in dfCollate.iloc[j,1]:
        for k in range(K):
        #当前需要之前没有，相当于放进去墨盒。
            model.addConstr(y[i,j,k] >= x[i,j,k] - x[i,j-1,k])

model.solve()
```

```
    %%%问题四

import pandas as pd
import numpy as np
import coptpy as copt
filePath="D:/workPlace/modelMath/data/附件4/"
file=filePath+'Ins1_5_10_2.xlsx'
# file=filePath+'Ins2_5_10_3.xlsx'
# file=filePath+'Ins3_7_10_2.xlsx'
# file=filePath+'Ins4_20_40_10.xlsx'
# 读取xlsx文件
```

```
# 读取xlsx文件
dfNum = pd.read_excel(file)
dfCollate = pd.read_excel(file, sheet_name=1)
dfClear = pd.read_excel(file, sheet_name=2)
dfCollate['所需墨盒编号'] = dfCollate['所需墨盒编号'].apply(lambda x: [i - 1 for i in eval(x)])
packageID=[]
inkID=[]
insertID=[]
for i in dfNum['包装种类编号'].dropna():
    packageID.append(i)
for i in dfNum['墨盒编号'].dropna():
    inkID.append(i)
for i in dfNum['插槽序号'].dropna():
    insertID.append(i)
I=len(insertID)
J=len(packageID)
K=len(inkID)
matrixTime=dfClear.iloc[:,1:].values.tolist()
# 初始化 COPT 环境和模型
env = copt.Envr()
model = env.createModel()

# 创建决策变量 x 和 s
x = model.addVars(I, J, K, vtype=copt.COPT.BINARY, nameprefix='x')
y = model.addVars(I, J, K, vtype=copt.COPT.BINARY, nameprefix='y')
z = model.addVars(J, J, vtype=copt.COPT.BINARY, nameprefix='z')
andZ = model.addVars(J, J, J, vtype=copt.COPT.BINARY, nameprefix='andZ')
#拿与放置的变量存储
TakeSet = model.addVars(I, J, K, K, vtype=copt.COPT.BINARY, nameprefix='TakeSet')

yu = model.addVars(J, J, I, J, K, K, vtype=copt.COPT.BINARY, nameprefix='yu')
# # 初始化变量初始值
# x_start = best_individual["x"]
# y_start = best_individual["y"]
# TakeSet_start = best_individual["TakeSet"]

# # 设置初始值
# model.start = {}
# for i in range(I):
#     for j in range(J):
#         for k in range(K):
#             model.start[x[i, j, k]] = x_start[i, j, k]
#             model.start[y[i, j, k]] = y_start[i, j, k]
#             for s in range(K):
#                 model.start[TakeSet[i, j, k, s]] = TakeSet_start[i, j, k, s]

```

```
#最小化y
model.setObjective(copt.quicksum(
    yu[q1,q2,i,j,t,s]*matrixTime[t][s] for q1 in range(J) for q2 in range(J) for j in range(1,J)
    for i in range(I) for t in range(K)for s in range(K)), copt.COPT.MINIMIZE)

#所有包装所需要颜色内必须要有
for j in range(J):
    for k in dfCollate.iloc[j,1]:
        model.addConstr(copt.quicksum(x[i, j, k] for i in range(I)) <= 1)

#q包装必须被打印
for q in range(J):
    model.addConstr(copt.quicksum(z[p,q]for p in range(J)) == 1)
#p轮必须打印东西
for p in range(J):
    model.addConstr(copt.quicksum(z[p,q]for q in range(J)) == 1)

#p轮必须打印东西
for q in range(J):
    for p in range(J):
        for k in dfCollate.iloc[q,1]:
            model.addConstr(z[p,q] <= copt.quicksum(x[i,p,k]for i in range(I)))

#指定包装顺序
for p in range(1,J):
    for q1 in range(J):
        for q2 in range(J):
            model.addConstr(andZ[p,q1,q2]>= z[p,q1]+z[p-1,q2]-1)
            model.addConstr(andZ[p,q1,q2]<= z[p,q1])
            model.addConstr(andZ[p,q1,q2]<= z[p-1,q2])


#最后结果and
for q1 in range(J):
    for q2 in range(J):
        for j in range(1,J):
            for i in range(I):
                for t in range(K):
                    for s in range(K):
                        model.addConstr(yu[q1,q2,i,j,t,s]>= andZ[j,q1,q2]+TakeSet[i,j,t,s]-1)
                        model.addConstr(yu[q1,q2,i,j,t,s]<= andZ[j,q1,q2])
                        model.addConstr(yu[q1,q2,i,j,t,s]<= TakeSet[i,j,t,s]-1)


#打印必须有一个
for j in range(J):
```

```
    for i in range(I):
        model.addConstr(copt.quicksum(x[i, j, k] for k in range(K)) == 1)


# y = model.addVars(I, J, K, vtype=copt.COPT.BINARY, nameprefix='y')
#识别拿去和放置
for j in range(1,J):
    for i in range(I):
        for t in range(K):
            for s in range(K):
                model.addConstr(TakeSet[i,j,t,s]>= y[i,j,s]+x[i,j-1,t]-1)
                model.addConstr(TakeSet[i,j,t,s]<= y[i,j,s])
                model.addConstr(TakeSet[i,j,t,s]<= x[i,j-1,t])

#设置顺序约束
# for p in range(J):
for j in range(J):
    temp=dfCollate.iloc[j,1]
    for k in range(1,len(temp)):
        model.addConstr(copt.quicksum((i*x[i,j,temp[k-1]]-i*x[i,j,temp[k]]) for i in range(I)) + 1 <=0)

#颜色判断变化
for j in range(1,J):
    for i in range(I):
#        for k in dfCollate.iloc[j,1]:
        for k in range(K):
        #当前需要之前没有，相当于放进去墨盒。
            model.addConstr(y[i,j,k] >= x[i,j,k] - x[i,j-1,k])

model.solve()
```

```
    %%%问题五

// CMakeProject1.cpp: 定义应用程序的入口点。
//
#include "gurobi_c++.h"
#include "CMakeProject.h"
#include "OpenXLSX/OpenXLSX.hpp"
#include "set"
#include "queue"
int packageNum = 0;
int inkNum = 0;
int slotNum = 0;
std::vector<std::vector<int>>inkIndexs;
std::vector<std::vector<int>>dist;
```

```cpp
std::vector<std::vector<int>>inkBeNeed;
std::vector<std::vector<int>>switchCost;
std::vector<int>slotInk;
std::vector<bool>slotValid;
std::vector<bool>isPrinted;
void initData(std::string filePath, std::string fileName);
int findSlotIsHaveInk(int ink);
bool isFinished();
int getBestNextPackaget();
int getInsertPos(int);
int ret = 1e9;
int main()
{

    std::string filePath = "C:\\Users\\LXJ\\Desktop\\data_B\\5\\";
    std::string fileName = "Ins1_5_5_2.xlsx";
    //fileName = "Ins2_10_50_15.xlsx";

    initData(filePath, fileName);

    try {
        // 创建一个Gurobi环境
        GRBEnv env = GRBEnv(true);
        env.start();

        // 创建一个新的模型
        GRBModel model = GRBModel(env);

        GRBVar w[20][20][20];
        GRBVar G[20][20][20];
        GRBVar z[20][20];
        GRBVar T[20];
        // 创建变量

        for (int i = 1; i <= packageNum; i++) {
            for (int j = 1; j <= slotNum; j++) {
                for (int k = 1; k <= inkNum; k++) {
                    std::string name = std::to_string(i) + "," + std::to_string(j) + "," + std::to_string(k);
                    w[i][j][k] = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "w" + name);
                    G[i][j][k] = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "G" + name);
                }
            }
        }
        for (int i = 1; i <= packageNum; i++) {
            for (int j = 1; j <= packageNum; j++) {
                std::string name = std::to_string(i) + "," + std::to_string(j);
```

```
            z[i][j] = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "z" + name);
        }
    }
    for (int i = 1; i <= packageNum; i++) {
        std::string name = std::to_string(i);
        T[i] = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "T" + name);
    }
    GRBLinExpr objective;
    for (int i = 1; i <= packageNum; i++) {
        objective += T[i];
    }

    model.setObjective(objective, GRB_MINIMIZE);
    // 设置目标函数

    for (int i = 1; i <= packageNum; i++) {
        for (int k = 1; k <= inkNum; k++) {
            GRBLinExpr objective;
            for (int j = 1; j <= slotNum; j++) {
                objective += w[i][j][k];
            }
            model.addConstr(objective <= 1);
        }
    }

    for (int i = 1; i <= packageNum; i++) {
        for (int j = 1; j <= slotNum; j++) {
            GRBLinExpr objective;
            for (int k = 1; k <= inkNum; k++) {
                objective += w[i][j][k];
            }
            model.addConstr(objective == 1);
        }

    }

    for (int p = 1; p <= packageNum; p++) {
        GRBLinExpr objective;
        for (int q = 1; q <= slotNum; q++) {
            objective += z[p][q];
        }
        model.addConstr(objective == 1);
    }

    for (int q = 1; q <= packageNum; q++) {
        GRBLinExpr objective;
```

```
            for (int p = 1; p <= packageNum; p++) {
                objective += z[p][q];
            }
            model.addConstr(objective == 1);
        }
        printf("runXXXXXXXXXXXXXXXXXXXXXX\n");
        for (int q = 1; q <= packageNum; q++) {
            for (int p = 1; p <= packageNum; p++) {
                for (int k = 0; k < (int)inkIndexs[q].size(); k++) {
                    GRBLinExpr objective1;
                    for (int j = 1; j <= slotNum; j++) {
                        objective1 += w[p][j][inkIndexs[q][k]];
                    }
                    objective1 = z[p][q] - objective1;
                    model.addConstr(objective1 <= 0);
                }
            }
        }
        //(6)

        for (int q = 1; q <= packageNum; q++) {
            for (int p = 1; p <= packageNum; p++) {
                for (int k = 0; k < (int)inkIndexs[q].size() - 1; k++) {
                    GRBQuadExpr objective1, objective2, objective3;
                    for (int j = 1; j <= slotNum; j++) {
                        objective1 += w[p][j][inkIndexs[q][k]] * j;
                        objective2 += w[p][j][inkIndexs[q][k + 1]] * j;
                    }
                    GRBVar tmp;
                    model.addConstr(tmp == objective1 - objective2);
                    objective3 = z[p][q] * tmp;
                    model.addConstr(objective3 <= 0);
                }
            }
        }
        printf("run!!!!!!!!!!!!!!!!!!!\n");
        for (int k = 1; k <= 2; k++) { //group
            GRBLinExpr objective;
            for (int i = 1; i <= packageNum; i++) {
                for (int j = 1; j <= slotNum; j++) {
                    GRBLinExpr objective;
                    for (int k = 1; k <= inkNum; k++) {
                        objective += G[i][j][k];
                    }
                    model.addConstr(objective == 1);
                }
```

```
            }
        }
        for (int i = 1; i <= packageNum - 1; i++) {
            for (int p = 1; p <= 2; p++) {
                for (int k1 = 1; k1 <= inkNum; k1++) {
                    for (int k2 = 1; k2 <= inkNum; k2++) {
                        GRBQuadExpr objective;
                        for (int j = 1; j <= slotNum; j++) {
                            GRBVar tmp;
                            model.addConstr(tmp == w[i][j][k1] * w[i + 1][j][k1]);
                            objective += G[i][j][p] * tmp * switchCost[k1][k2];
                        }
                        model.addConstr(T[i] >= objective);
                    }
                }
            }
        }

        model.optimize();

        if (model.get(GRB_IntAttr_Status) == GRB_OPTIMAL) {
            std::cout << "最优解:" << std::endl;
            for (int i = 1; i <= slotNum; i++) {
                for (int j = 1; j <= packageNum; j++) {
                    for (auto k : inkIndexs[j]) {
                        std::cout << w[i][j][k].get(GRB_DoubleAttr_X) << std::endl;
                    }
                }
            }
            std::cout << "最小化目标值: " << model.get(GRB_DoubleAttr_ObjVal) << std::endl;
        }
        else {
            std::cout << "未找到最优解" << std::endl;
        }
    }
    catch (GRBException e) {
        std::cout << "Gurobi 错误: " << e.getErrorCode() << " " << e.getMessage() << std::endl;
    }
    catch (...) {
        std::cout << "遇到未知错误" << std::endl;
    }
    return 0;

}
//找到插槽内是否有这个墨盒
int findSlotIsHaveInk(int ink) {
```

```
    for (int i = 1; i <= slotNum; i++) {
        if (slotInk[i] == ink)return i;
    }
    return -1;
}
//找到墨盒应该被放入的位置
int getInsertPos(int ink)
{
    int insertPos = 0;
    for (int i = 1; i <= slotNum; i++) {
        if (slotValid[i])continue;
        if (slotInk[i] == 0) {
            insertPos = i;
        }
    }
    if (insertPos != 0)return insertPos;
    for (int i = 1; i <= slotNum; i++) {
        if (slotValid[i])continue;
        if (slotInk[i] != ink) {
            insertPos = i;
        }
    }
    return insertPos;
}
//找到下一个被打印的物品
int getBestNextPackaget()
{
    int ret = -1;
    int minCost = 1e9;
    for (int i = 1; i <= packageNum; i++) {
        if (isPrinted[i])continue;
        int cost = 0;
        for (int j = 0; j < inkIndexs[i].size(); j++) {
            if (findSlotIsHaveInk(inkIndexs[i][j]) == -1) {
                cost++;
            }
        }
        if (cost < minCost) {
            ret = i;
            minCost = cost;
        }
    }
    return ret;
}
//是否结束
bool isFinished() {
```

```
245     for (int i = 1; i <= packageNum; i++) {
246         if (isPrinted[i] == false)return false;
247     }
248     return true;
249 }
250 //初始化数据
251 void initData(std::string filePath, std::string fileName)
252 {
253 
254     int num[4];
255     int x = 0;
256     int i = 0;
257     for (int j = 0; j < fileName.length(); j++) {
258         if (fileName[j] >= '0' && fileName[j] <= '9') {
259             x = x * 10 + fileName[j] - '0';
260         }
261         else {
262             if (x == 0)continue;
263             num[i++] = x;
264             x = 0;
265         }
266     }
267     packageNum = num[1];
268     inkNum = num[2];
269     slotNum = num[3];
270     // 创建一个文档对象并打开xlsx文件
271     OpenXLSX::XLDocument doc;
272     printf("read over1!");
273     doc.open(filePath + fileName);
274     printf("read over2!");
275     // 获取第一个工作表
276     auto wks = doc.workbook().worksheet("sheet1");
277     printf("read over3!");
278 
279     i = 0;
280     inkIndexs.emplace_back(std::vector<int>());
281     for (const auto& row : wks.rows()) {
282         if (++i == 1)continue;
283         std::string cell_address = OpenXLSX::XLCellReference(i, 2).address();
284         auto cell = wks.cell(cell_address);
285         std::string str = cell.value().get<std::string>();
286         int index = 0;
287         std::vector<int>tmp;
288         for (int j = 0; j < str.length(); j++) {
289             if (str[j] >= '0' && str[j] <= '9') {
290                 index = index * 10 + str[j] - '0';
```

```
            }
            else {
                if (index == 0)continue;
                tmp.emplace_back(index);
                index = 0;
            }
        }
        inkIndexs.emplace_back(tmp);
    }

    wks = doc.workbook().worksheet("sheet2");
    switchCost.resize(inkNum + 1);
    for (int i = 0; i <= inkNum; i++) {
        switchCost[i].resize(inkNum + 1, 0);
    }
    for (int i = 1; i <= inkNum; i++) {
        for (int j = 1; j <= inkNum; j++) {
            std::string cell_address = OpenXLSX::XLCellReference(i + 1, j + 1).address();
            auto cell = wks.cell(cell_address);
            switchCost[i][j] = cell.value().get<int>();
        }
    }
    doc.close();
}
```