中国研究生创新实践系列大赛

中国光谷·“华为杯”第十九届中国研究生数学建模竞赛

学　　校	杭州电子科技大学
参赛队号	22103360818
队员姓名	1.曹旭涛
	2.洪江南
	3.胡瑾

中国研究生创新实践系列大赛

中国光谷·“华为杯”第十九届中国研究生数学建模竞赛

题　目　　基于整数规划的组批和下料优化问题研究

摘　　要：

随着制造业生产技术的不断进步，企业通常采用“订单组批+批量生产+订单分拣”的模式进行生产，在该生产模式中，订单组批与下料优化至关重要，两者能降低企业的生产成本，减少资源的浪费。与此同时，订单组批与下料优化问题具有规则复杂、目标精细的特点。基于以上背景，本文将复杂的订单组批与下料优化问题拆解为多个步骤，由浅入深，通过进行整数规划并设计启发式价值修正算法逐步进行求解，最终取得全局较优解。

针对问题一，依据题意对目标函数和约束条件进行分析，据此进行整数规划，首先对问题一的数据集进行分析，定位所求模型属于三阶段约束排样方式(3E)，紧接着对排样算法进行优化设计，在排样具体过程中，本模型采用自底向上的构建方式，并使用**基于贪心策略的迭代顺序修正算法**进行优化。首先对数据集进行预处理，初始化 item 的价值为 item 的高度，并按价值对 item 进行排序，若价值相同则按照 item 的宽度从大到小排序，对得到的所有 item 序列输入基于贪心的顺序启发式算法，算法的核心思想在于尽可能多的利用每一块板材的余料空间进行排样；长度或宽度相同的 item 尽可能的叠加在一个 stack 上；形状相同或相似的 item 尽可能在一块板材上；先构建多个 stack 组合成 strip，多个 strip 构建成每一个排样。算法运行至所有 item 序列全部切割完成，保存当前排样结果，之后算法将基于每个 item 原价值和所在板材的利用率更新价值，并按照价值对 item 重新进行排序，然后基于此 item 序列生成新的排样方案，并迭代 N 轮，保存最佳的排样结果，最终求得数据集 **dataA1** 板材利用率为 **93.87%(**板材用量 **89)**，运行时间 **0.24s**，**dataA2** 板材利用率为 **92.08%(**板材用量 **90)**，运行时间 **0.22s**，**dataA3** 板材利用率为 **94.08%(**板材用量 **89)**，运行时间 **0.24s**，**dataA4** 的板材利用率为 **94.08%(**板材用量 **87)**，运行时间 **0.28s**。

针对问题二，本文首先结合组批的概念，在问题一的整数规划的基础上加入与批次相关的约束，而目标函数不变；紧接着对组批算法规则进行优化设计，整体上首先需要将 item 根据订单号进行分组，之后进行订单组批，在组批算法中，模型首先依靠每个订单中 item 的材质和数量对订单进行**特征编码**，输入基于**杰卡德距离**的订单**凝聚层次聚类算法**。聚类过程需在满足约束条件的同时，让尽可能多的订单聚成一类。该聚类算法会将相似型号材质的订单尽量放在同一批次，以减少板材的用量。得到聚类后的订单组批后，基于每个批次中所有订单的 item，对其按照材质进行分组；然后将每一个分组输入问题一模型进行独立排样，最终求得数据集 **dataB1** 的板材利用率为 **77.10%(板材用量 3863)**，运行时间 **18.73s**，**dataB2** 的板材利用率为 **75.56%(**板材用量 **2550)**，运行时间 **7.76s**，**dataB3** 的板材利用率为 **74.69%(**板材用量 **2589)**，运行时间 **7.70s**，**dataB4** 的板材利用率为 **76.15%(**板材用量 **2666)**，运行时间 **6.62s**，**dataB5** 板材利用率为 **74.94%(**板材用量 **4123)**，运行时间 **23.78s**。

最后，本文对算法的有效性、稳定性和复杂度进行了评估分析，得到问题一算法的时间复杂度约为$O(n^3)$，问题二算法的时间复杂度约为$O(n^3)$。

关键词：整数规划　启发式价值修正算法　凝聚层次聚类算法　组批下料优化

目录

一、 前言

1.1 研究背景

随着制造业生产技术的不断进步，“个性化定制生产”应运而生。“个性化定制生产”使企业面临着“产品种类繁多但批量较小”的问题，为解决这一问题，企业通常采用“订单组批+批量生产+订单分拣”的模式进行生产。

在该生产模式中，订单组批与下料优化至关重要。订单组批是将不同订单重新组合，在保证按时交货的前提下，将原材料相同、生产工艺相似的产品尽可能地同批次生产，以此提高设备的生产效率和原材料的利用率。

下料优化问题在制造业的生产加工过程中广泛存在：服装制造业中布匹皮革的分割、木材加工业中圆木的分割、机械制造业中金属板的剪切，都属于不同类型的下料问题。下料优化本质上是寻找一种方案，实现给定的几何图形上不重叠的放入更多的几何图形，使原材料利用率最高[1]。当前，全球性的能源问题日益凸显，下料作为众多制造企业生产链中产品及零部件生产的第一道工序，消耗的材料和资源不容小视。优化下料方案，提高材料利用率，不仅能降低企业的生产成本，更能减少资源和能源的浪费，实现可持续发展。

1.2 问题重述

方形件产品是以板材（Bin）为主要原片、通过平面加工后的几种板式配件装配而形成的一类产品。合理规划方形件在板材上的布局，既能简化切割过程，又能减少下料过程中存在的板材浪费。切割过程中有关概念说明如下：

1）齐头切是一种沿直线从板材一边贯穿到对边的切割方式，每次切割后板材分离为两块。

2）三阶段排样方式：同一阶段的切割方向相同，相邻阶段的切割方向垂直。在三个阶段内将板材切割成 Item，即先切割成 Stripe，再一步切割成 Stack，最终切割成 Item。切割步骤如图 1-1 所示。

3）Item（产品项）：最终切割形成的模块。

4）Stack（栈）：由一个或多个 Item 相邻排列组成。

5）Stripe（条带）：由一个或多个 Stack 相邻排列组成。

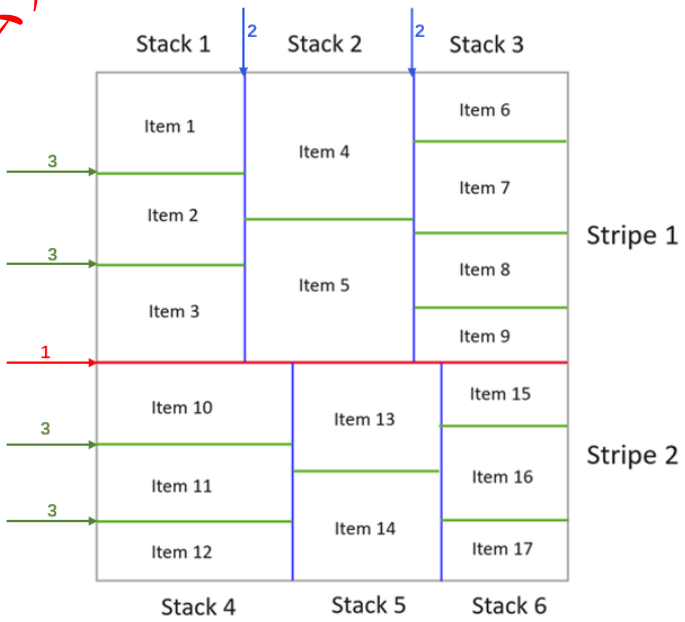

图 1-1 三阶段切割图示

基于数据集 A、数据集 B 中方形件产品的材质、需求数量及尺寸规格，本文研究矩形原材料上方形件产品的排样优化及订单组批问题：

问题一：排样优化问题

本题以数据集 A 为输入，建立整数规划模型，对方形件在板材上的排样方式进行优化，在满足订单需求的条件下，使板材用量尽可能少。本题要求相同栈（stack）里的产品项（item）的宽度（或长度）相同，且最终切割生成的产品项是完整的，非拼接而成。

问题二：订单组批问题

本题以数据集 B 为输入，建立整数规划模型，首先对全部订单中的产品根据材质重新组批，进一步地对每个批次分别进行排样优化，在满足订单需求的条件下，实现板材用量尽可能少。在问题一的基础上，本题进一步要求：

1）每份订单当且仅当出现在一个批次中；

2）每个批次中的相同材质的产品项（Item）才能使用同一块板材原片进行排样；

3）为保证加工环节快速流转，每个批次产品项（Item）总数不能超过限定值，单个批次产品项总数上限为 1000；

4）因工厂产能限制，每个批次产品项（Item）的面积总和不能超过限定值，单个批次产品项的面积总和上限为 250m^2.

二、 模型假设与符号说明

2.1 模型假设

假设 1：本题只考虑齐头切的切割方式；

假设 2：切割阶段数不超过 3，切割方向为第一阶段水平切割，第二阶段垂直切割，第三阶段水平切割；

假设 3：排样方式为精确排样；

假设 4：板材原片仅有一种规格且数量充足，原片长度 L 为 2440mm，原片宽度 W 为 120mm；

假设 5：排样方案不用考虑锯缝宽度（即切割的缝隙宽度）影响。

2.2 符号说明

本节对文中出现的主要变量及参数进行说明：

表 2-1 主要变量及参数说明

模型符号	符号说明
i	产品项（Item）的 ID
n	单个数据集中产品项总数
j	Stack 的标记
k	Strip 的标记
m	板材原片（Bin）的标记
x	订单（Order）的标记
y	批次（Batch）的标记

z	订单总数
l_i	产品项 i 的长度
w_i	产品项 i 的宽度
t_i	产品项 i 的材质
L	板材原片的长度
W	板材原片的宽度
RL	Stack/Stripe/Bin 的剩余长度
RW	Stack/Stripe/Bin 的剩余宽度
$\alpha_{j,i}$	识别产品项 i 是否在 Stack j 中排样
$\beta_{k,j}$	识别 Stack j 是否在 Stripe k 中排样
$\gamma_{m,k}$	识别 Stripe k 是否在板材 m 中排样
$\delta_{m,i,j}$	识别产品项是否 i 包含在 Stack j 中，Stack j 是否包含在 Stripe j 中，Stripe j 是否包含在 Bin m 中
v_{i0}	产品项 i 的初始价值
P_T	第 T 次迭代生成的排样方案
v_{Tt}	第 T 次迭代修正后产品项 i 的价值
u_T	第 T 次排样的板材利用率

三、 问题一的模型建立与求解

3.1 问题分析

针对问题一，问题要求给出方形件的排样优化策略方案，期望是尽可能减少板材原片用量，对目标函数和约束条件进行分析，据此建立整数规划模型进行求解，而该问题的优化目标主要体现在对板材的利用率上。

首先对问题一的数据集进行分析，发现所有 item 的数量都为 1 且均为相同的材质，并且题目约束排样方式为精确排样，因此模型属于三阶段匀质排样方式(3E)，问题一的核心和难点就在于如何构建符合 3E 的且板材利用率高的排样算法。在排样具体过程中，本模型采用自底向上的构建方式，并使用迭代顺序启发式价值修正的方法，问题一的排样优化模型整体思路如图 3-1 所示：

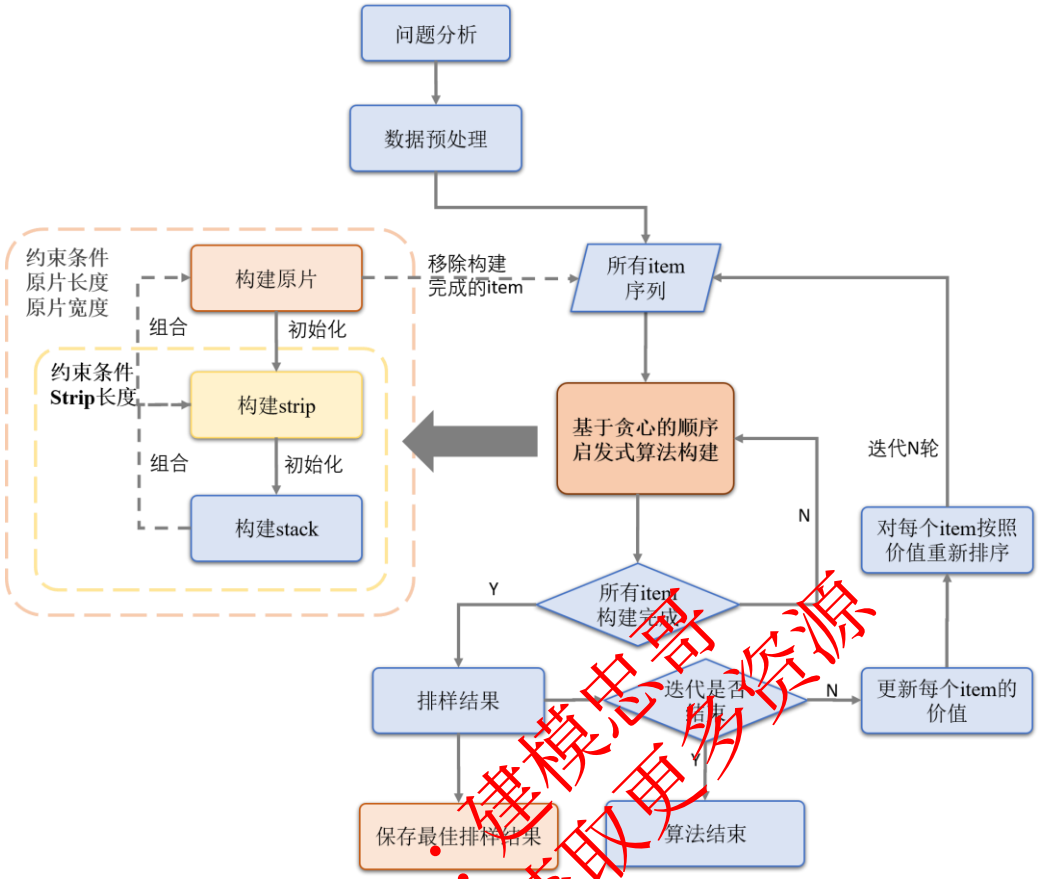

图 3-1 问题一求解流程图

首先模型对数据集进行预处理，初始化 item 的价值，并按价值对 item 进行排序，对得到的所有 item 序列输入基于贪心的顺序启发式算法，算法的核心思想在于尽可能多的利用每一块板材的余料空间进行排样；长度或宽度相同的 item 尽可能的叠加在一个 stack 上；形状相同或相似的 item 尽可能在一块板材上；先构建 stack 组合成 strip，每一个组合成的 strip 构建成每个板材原片的排样，算法运行至所有 item 序列全部切割完成，保存当前排样结果(板材利用率)，之后算法将根据每个 item 的板材利用率和原价值进行更新，并按照价值对 item 重新进行排序，然后基于此 item 序列重新进行排样，并迭代 N 轮，保存最佳的排样结果(板材利用率最高)。

3.2 模型预处理

首先，定义 Item 的长边为长（length）,短边为宽（width）。本节仅考虑长边方向上的排样，即排样结果中产品项的长边与板材的长边平行，不考虑产品项长边与板材长边垂直的情况，即不考虑产品项旋转 90° 后的排样方案。

进一步地，将各数据集中的 Item（产品项）按其长边由大到小排序，再给出 Stack（栈）、Stripe（条带）、Bin（板材原片）的标记方法，其中，Stack 的标记对应其包含的最长 Item 的 ID；Stripe 的标记对应其包含的最长 Stack 的标记；Bin 的标记对应其包含的最长的 Stack 的标记。

以图 3-2 为例进行说明，产品项 i（i 为 item_id）的长度记为 l_i，图中产品项的长度排序为 $l_{263}>l_{36}>l_3>l_{161}>l_{485}>l_{487}$，红线表示第一阶段的切割方向，切割生成的模块为 Stripe，图中 Stripe 的标记为 Stripe263、Stripe3；蓝线表示第二阶段的切割方向，切割生成的模块为 Stack，图中 Stack3、Stack161、Stack263、Stack36 均仅包含一个 Item，Stack485 包含两个 Item，用较长 Item485 的 ID 作为该 Stack 的标记；绿线表示第三阶段的切割方向，切割

生成的最终模块为 Item。图中板材标记为 Bin263。

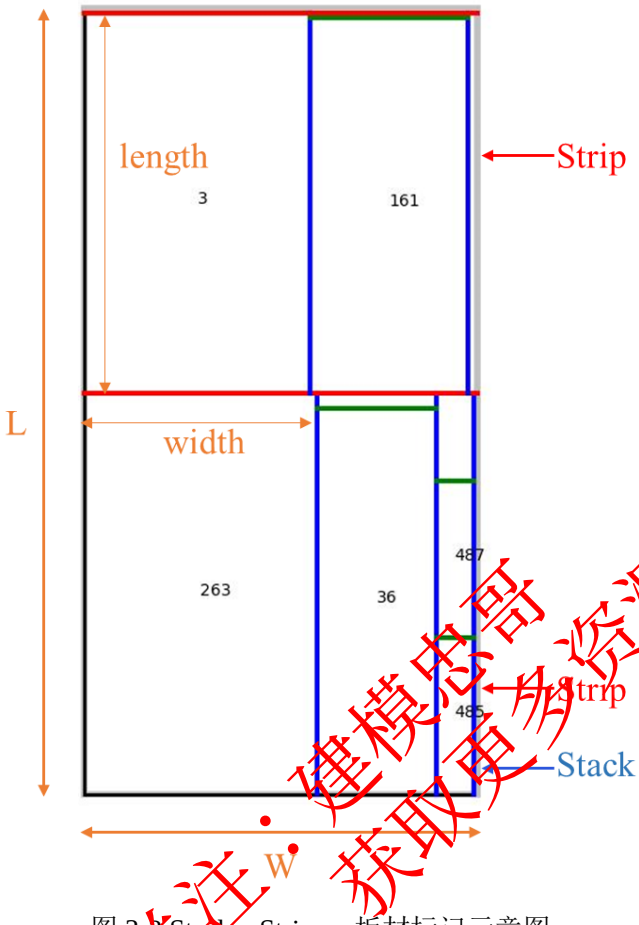

图 3-2 Stack、Stripe、板材标记示意图

3.3 模型建立

排样问题的常见求解方法有线性规划法、启发式搜索算法等，线性规划用于求解线性约束下线性目标函数的最值问题。排样问题中线性规划的决策变量是排样方式，即产品项 i 在板材 m 中是否排样。当求解结果不为整数时，向上或向下取整会偏离最优解，向上取整可能会导致板材浪费，向下取整可能无法满足订单需求。本文将线性规划模型中的部分变量限制为整数值，建立整数线性规划模型，求解最优排样方案。

3.3.1 决策变量

排样问题的决策变量用于判断产品项 i 是否在板材 m 中排样，可细分为 4 个问题：

（1）当且仅当产品项 i 包含在 Stack j 中，$\alpha_{j,i}$ 取 1.

$$\alpha_{j,i} \in \{0,1\},\quad j=1,\ldots,n,\quad i=j,\ldots,n \tag{3-1}$$

（2）当且仅当 Stack j 包含在 Stripe k 中，$\beta_{k,j}$ 取 1.

$$\beta_{k,j} \in \{0,1\},\quad k=1,\ldots,n,\quad j=k,\ldots,n \tag{3-2}$$

（3）当且仅当 Stripe k 包含在板材原片 Bin m 中，$\gamma_{m,k}$ 取 1.

$$\gamma_{m,k} \in \{0,1\},\quad m=1,\ldots,n,\quad k=m,\ldots,n \tag{3-3}$$

（4）为计算 Stripe 的长度，定义变量 $\delta_{m,i,j}$：

$$\delta_{m,i,j} \in \{0,1\},\ m=1,\ldots,n-1,\ i=m+1,\ldots,n,\ \text{and}\ j=m,\ldots,i-1 \tag{3-4}$$

当且仅当产品项 i 包含在 Stack j 中，Stack j 包含在 Stripe j 中，Stripe j 包含在 Bin m 中时，$\delta_{m,i,j}$ 取 1，即：

$$\delta_{m,i,j}=1 \Leftrightarrow \alpha_{j,i}=1 \cap \gamma_{m,j}=1 \tag{3-5}$$

3.3.2 目标函数

为降低生产成本和资源消耗，排样问题的优化目标是使板材用料 Z 尽可能少，即

$$Z=\min\sum_{i=1}^{n}\gamma_{m,m} \tag{3-6}$$

我们假定排样时每条 Stripe 中最长的 Stack 均由单个 Item 组成，即 Strip 的长度由其包含的 Stack 中最长 Item 的长度确定，该假定有助于确定条带长度。

3.3.3 约束条件

为满足订单需求，保证所有产品均被生产，考虑到板材的长度及宽度限制，做出如下约束：

（1）保证每个产品项 i 仅排样一次，如式（3-5）所示，确保排样不重不漏。

$$\sum_{j=1}^{i}\alpha_{j,i}=1,\forall i=1,\ldots,n \tag{3-7}$$

（2）只有宽度（w）相同的产品项 i、j，才能包含在同一段 Stack 中，且需保证产品项长度之和不超过板材的长度，否则 $\alpha_{j,i}=0$。

$$\sum_{i=j+1}^{n}\alpha_{j,i}\leqslant(n-j)\alpha_{j,j},\quad \forall j=1,\ldots n-1, \tag{3-8}$$

$$\alpha_{j,i}=0,\ \forall\ w_i\neq w_j \cup l_i+l_j>L,\ \text{其中}i>j,\ j=1,\ldots,n-1 \tag{3-9}$$

（3）每段 Stack j 仅包含在 1 条 Stripe k 中:

$$\sum_{k=1}^{j}\beta_{k,j}=\alpha_{j,j},\quad \forall j=1,\ldots,n \tag{3-10}$$

（4）Stack 的长度不应超过其所在 Stripe 的长度：单个 Stcak 的长度由其包含的产品项长度之和确定，如式（3-10）左边部分；Stripe 的长度由其包含的最长的 Stack 长度确定。

为避免具有相同长度的 Stack 在标记时引起混淆，我们用“严格小于”和“小于等于”进行区分：当一个 Stripe 中出现最大长度相等的多条 Stack 时，我们规定用标号较小的 Stack 标记该条 Stripe.

$$\sum_{i=j}^{n}l_i\alpha_{j,i}<\sum_{i=k}^{n}l_i\alpha_{k,i}+(L+1)(1-\beta_{k,j}),\quad \forall k=2,\ldots,n,\quad \forall j=1,\ldots,k-1, \tag{3-11}$$

$$\sum_{i=j}^{n} l_i \alpha_{j,i} \leqslant \sum_{i=k}^{n} l_i \alpha_{k,i} + L(1-\beta_{k,j}),\ \ \forall k = 1,\ldots,n-1,\ \ \forall j = k+1,\ldots,n, \tag{3-12}$$

（5）在宽度方向上，同一 Strip 中 Stack 的宽度之和不应超过板材原片的宽度 W。

$$\sum_{j=k}^{n} w_j \beta_{k,j} \leqslant W \beta_{k,k},\ \ \forall k = 1,\ldots,n-1 \tag{3-13}$$

（6）所有 Stripe 的长度之和 $\sum_{i=m}^{n} l_i \gamma_{m,i} + \sum_{i=m+1}^{n} l_i \sum_{j=m}^{i-1} \delta_{m,i,j}$ 不应超过板材原片的长度 L。

$$\sum_{m=1}^{k} \gamma_{m,k} = \beta_{k,k},\quad \forall k = 1,\ldots,n \tag{3-14}$$

$$\sum_{i=m}^{n} l_i \gamma_{m,i} + \sum_{i=m+1}^{n} l_i \sum_{j=m}^{i-1} \delta_{m,i,j} \leqslant L\gamma_{m,m},\ \ \forall m = 1,\ldots,n-1 \tag{3-15}$$

$$\begin{aligned} &\alpha_{j,i} + \gamma_{m,j} - 1 \leqslant \delta_{m,i,j} \leqslant (\alpha_{j,i} + \gamma_{m,j})/2, \\ &\forall m = 1,\ldots,n-1,\ \forall i = m+1,\ldots,n,\ j = m,\ldots,i-1 \end{aligned} \tag{3-16}$$

（7）保证所有 Stripe 均被排样：

$$\sum_{k=m+1}^{n} \gamma_{m,k} \leqslant (n-1)\gamma_{m,m},\ \ \forall m = 1,\ldots,n-1 \tag{3-17}$$

综上，问题一建立的整数线性规划模型总结如下：

$$Z=\min\sum_{i=1}^{n}\gamma_{m,m}$$

$$s.t.\begin{cases}\sum_{j=1}^{i}\alpha_{j,i}=1,\forall i=1,\ldots,n\\ \sum_{i=j+1}^{n}\alpha_{j,i}\leqslant(n-j)\alpha_{j,j},\ \ \forall j=1,\ldots n-1,\\ \alpha_{j,i}=0,\ \ \forall\ w_i\neq w_j\cup l_i+l_j>L,\ \text{其中}i>j,\ \ j=1,\ldots,n-1\\ \sum_{k=1}^{j}\beta_{k,j}=\alpha_{j,j},\quad \forall j=1,\ldots,n\\ \sum_{i=j}^{n}l_i\alpha_{j,i}<\sum_{i=k}^{n}l_i\alpha_{k,i}+(L+1)(1-\beta_{k,j}),\ \ \forall k=2,\ldots,n,\ \ \forall j=1,\ldots,k-1,\\ \sum_{i=j}^{n}l_i\alpha_{j,i}\leqslant\sum_{i=k}^{n}l_i\alpha_{k,i}+L(1-\beta_{k,j}),\ \ \forall k=1,\ldots,n-1,\ \ \forall j=k+1,\ldots,n,\\ \sum_{j=k}^{n}w_j\beta_{k,j}\leqslant W\beta_{k,k},\ \ \forall k=1,\ldots,n-1\\ \sum_{m=1}^{k}\gamma_{m,k}=\beta_{k,k},\quad \forall k=1,\ldots,n\\ \sum_{i=m}^{n}l_i\gamma_{m,i}+\sum_{i=m+1}^{n}l_i\sum_{j=m}^{i-1}\delta_{m,i,j}\leqslant L\gamma_{m,m},\ \ \forall m=1,\ldots,n-1\\ \alpha_{j,i}+\gamma_{m,j}-1\leqslant\delta_{m,i,j}\leqslant(\alpha_{j,i}+\gamma_{m,j})/2,\\ \qquad\forall m=1,\ldots,n-1,\ \forall i=m+1,\ldots,n,\ j=m,\ldots,i-1\\ \sum_{k=m+1}^{n}\gamma_{m,k}\leqslant(n-1)\gamma_{m,m},\ \ \forall m=1,\ldots,n-1\end{cases}\tag{3-18}$$

3.4 模型求解

对于整数线性规划模型的求解，通常利用商业优化器，如 CPLEX 优化器，该优化器在求解时运算量较大，只能精确求解小规模问题，对大规模问题的计算时间较长，且结果并不一定是实际需要的方案。

本题以板材用料尽可能少为目标，采用**基于贪心策略的迭代顺序修正算法**求解排样优化问题。考虑到排样设计的难易程度和板材的消耗量，我们首先采用**贪心策略、自底向上**逐级排样，然后放宽排样规则，考虑**产品项旋转策略**，最后引入“产品项价值”，基于**价值修正策略**调整产品项顺序，不断迭代生成新的排样方案，从中选择最优方案。

算法核心思想如表 3-1 所示：

表 3-1 问题一算法核心思想

基于贪心策略的迭代顺序修正算法
（1）**贪心策略** 我们在首次排样时，总是优先考虑长度或宽度较大的产品项，因为此类产品较难与其他产品组合，通常需要新建 Stack 或 Stripe。如果将此类产品留在排样的靠后阶段，可能缺少尺寸较小的产品项与之组合，从而使板材上出现大量空缺，不利于减少板材用量。
（2）**自底向上排样策略** 在遍历搜索时，我们优先考虑 Item 能否排入当前存在剩余容量的 Stack，不满足时再考虑新建 Stack，进一步考虑新建 Stripe。这样做可以保证每段 Stack、

每条 Stripe 都被充分排样，使排样空隙尽可能少。

（3）**产品项旋转策略** 如果仅考虑长度方向上的排样方案，将不利于长条形 Item 的排样，因其长度较长，通常无法在已有 Stack 的剩余部分排样，甚至难以在已有 Stripe 的剩余部分排样，此时如果新建 Stripe 或新建板材，将会产生大量排样空隙。旋转策略将在一定程度上改善这一问题，在 Item 的长度无法满足条件是时，考虑能否在宽度方向上进行排样。

（4）**产品价值修正策略** 本题在首次排样时，以长度作为产品项的排序标准，此时长度和宽度相近的产品项的顺序相连，这会导致在下一次搜索时，第一个搜索到的产品项与刚排入的产品项尺寸相近，较难组合，此时我们希望尽快搜索到尺寸较少的产品项，与先前产品项进行组合。基于材料利用率，我们计算产品价值，并不断进行修正，使尺寸较大的产品项与尺寸较小的产品项交替排列，降低搜索次数，使算法耗时更短，排样方案更优。

3.4.1 算法主框架

本节首先介绍算法的主体框架，其中涉及到的策略和子算法在后续详细介绍。

算法 1　基于贪心策略的迭代顺序修正算法

输入：　产品项数据集
输出：　最优排样方案
Step1:　令 T=1，Z=∞，
　　按长度对产品项排序，得到产品项初始序列 A，基于长度设定产品项初始价值 v_0
Step2:　生成当前排样方案
　　Step2.1: 令 PT=∅，调用贪心算法进行排样，已排样 Item 移出序列 A：
　　当贪心算法在长度方向上无法继续排样时，考虑旋转策略
　　Step2.2: 若序列 A≠∅，返回 Step2.1；否则转 Step3
Step3: 若 Z 为当前最小值，保留当前排样方案 P_T，并更新 Z；
Step4: 调用价值修正算子，修正产品项价值，并根据修正价值对产品项重新排序，更新序列 A
Step5：令 T=T+1，若 T＞T_{max}，算法终止，输出最优排样方案，否则转 Step2

其中，T 为当前迭代次数，T_{max} 为设定的迭代次数，Z 为板材用量，P_T 为第 T 次迭代生成的排样方案。

问题一算法流程如图 3-3 所示：

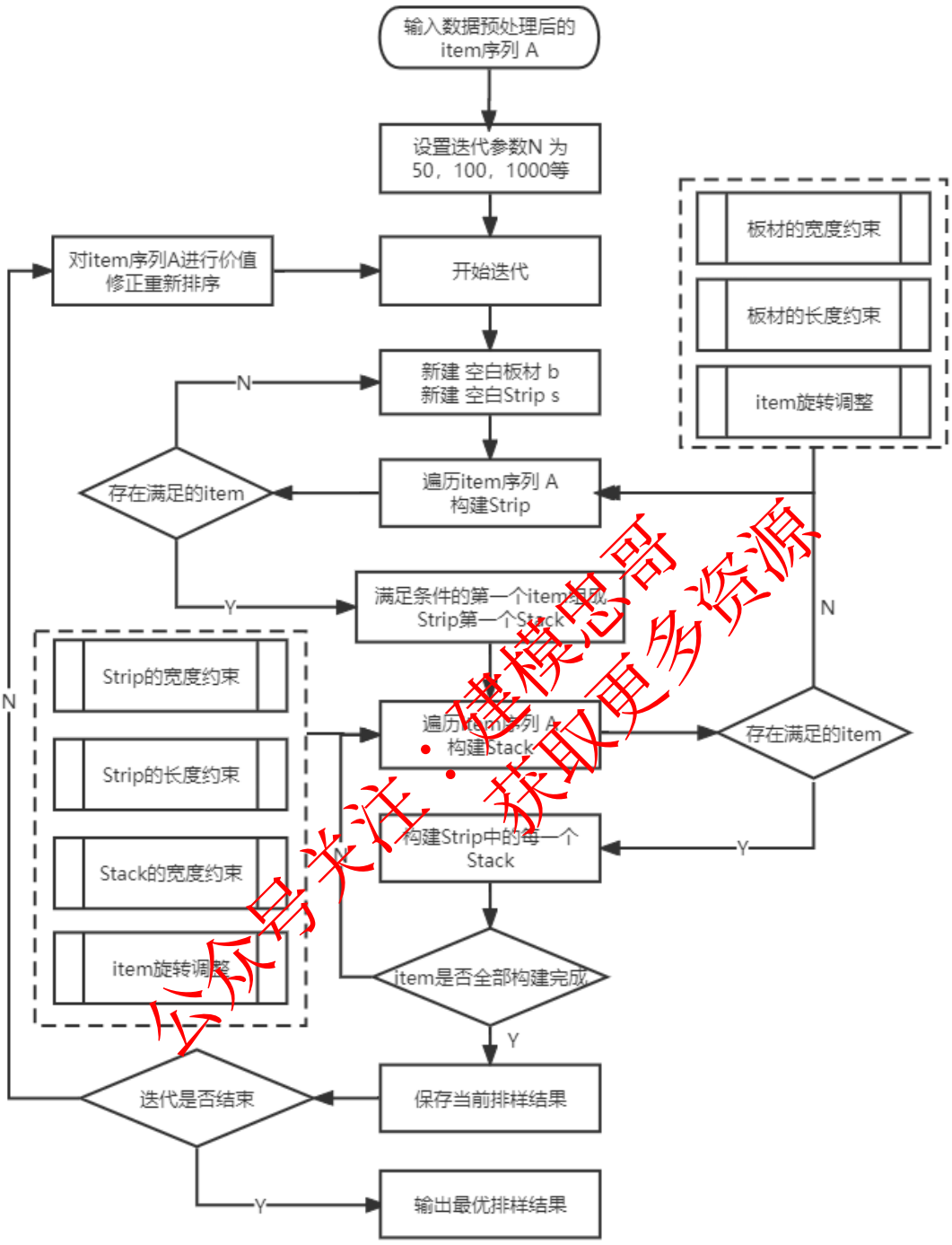

图 3-3 问题一算法流程图

3.4.2 贪心算法

“贪心”指在排样时总是按照当前产品序列进行搜索，选择最先满足条件的产品项进行排样。初次排样时，在满足约束的条件下，总是选择长度较大的产品项优先排样；当产品价值更新后，在满足约束的条件下，总是选择价值较大的产品项优先排样。贪心算法实现步骤如下所示：

算法 2 贪心算法

输入：产品项序列 A

输出：当前排样方案

Step1：构建板材。新建一块空白板材 B；

Step2：构建 Stripe。在板材 B 中新建一个空白 Stripe，记为 s；

Step2.1：遍历序列 A 中所有 Item，当 Item i 的长度和宽度不超过当前板材中剩余的长度和宽度，即 $l_i \le RL_{\text{bin\_b}} \cap w_i \le RW_{\text{bin\_b}}$，将最先满足该条件的 Item a 放入 s 中，同时从序列 A 中移除 Item a。此时 Item a 单独组成 Stack a， Stripe s 的最大长度由 Item a 的长度 l_a 确定。

Step2.2：若所有 Item 的长和宽均不满足条件，则返回 Step2；

Step3：构建 Stack。

Step3.1：遍历序列 A 中剩余 Item，当 Item i 的长度不超过 Stripe s 的长度、宽度不超过 Stripe s 的剩余宽度，即 $l_i \le l_a \cap w_i \le RW_{sp\_s}$，将最先满足条件的 Item b 放入 Stripe s，同时从序列 A 中移除 Item b。Item b 生成 Stack b，Stack b 的最大长度为 l_a。若所有剩余 Item 的长和宽均不满足条件，则返回 Step3。

Step3.2：继续遍历序列 A 中剩余 Item，当 Item i 的长度不超过 Stack b 的剩余长度、宽度等于 Item$b$ 的宽度，即 $l_i \le RL_{stk\_b} \cap w_i = w_b$，将最先满足条件的的 Item c 放入 Stack b，同时从序列 A 中移除 Item c。若所有剩余 Item 的长和宽均不满足条件，则返回 Step3.1。若存在满足条件的 Item，重复 Step3.2。

3.4.3 产品项旋转策略

在贪心算法中，我们仅考虑了产品项在长度方向上的排样，在这种排样方式下，当产品项 i 的长度大于当前 Stack 或当前 Stripe 中的剩余长度时，算法将跳过产品项 i，继续搜索满足长度条件的产品项进行排样。如果此时将产品项 i 进行 90° 旋转，若旋转后产品项 i 的短边不超过当前 Stack 的剩余长度，同时长边与当前 Stack 的宽度相等，则可将产品项 i 排入当前 Stack。

通过修正决策变量 $\alpha_{r,j,i}$ 来区分产品项是否进行旋转：

$$\alpha_{r,j,i} \in \{0,1\}$$
$$\begin{cases} \alpha_{1j,i} = 1, \text{ 产品项}i\text{旋转后在Stack } j \text{ 中排样} \\ \alpha_{0j,i} = 1, \text{ 产品项}i\text{未进行旋转，并在Stack } j \text{ 中排样} \end{cases} \tag{3-19}$$

（2）算法实现

考虑产品项 i 可旋转时，对前述算法中生成 Stripe、生成 Stack、组建 Stack、的步骤进行修正：

在算法 2 节 Step2.1 中，当序列 A 中剩余 Item 的长度均超过当前板材的剩余长度时，我们对剩余 Item 按宽度重新排序，遍历排序后的所有 Item，当出现 Item 的宽度小于当前板材的剩余长度、Item 的长度小于当前板材的剩余宽度时，旋转该 Item，将该 Item 放置

在当前板材中，生成新的 Stripe，并移出序列 A。

在算法 2 Step3.1 中，当序列 A 中剩余 Item 的长度均超过当前 Stripe 的最大长度时，我们对剩余 Item 按宽度重新排序，遍历排序后的所有 Item，当出现 Item 的宽度小于当前 Stripe 的最大长度、Item 的长度小于当前 Stripe 的剩余宽度时，旋转该 Item，生成新的 Stack，并将其紧贴已有 Stack 进行排样。

在算法 2 Step3.2 中，当序列 A 中剩余 Item 的长度均超过当前 Stack 剩余长度时，我们对剩余 Item 按宽度重新排序，遍历排序后的所有 Item，当出现 Item 的长度等于当前 Stack 的宽度、Item 的宽度小于当前 Stack 的剩余长度时，旋转该 Item，将该 Item 叠放在当前 Stack 中，并移出序列 A。

旋转优化的实现结果如图 3-4 所示：

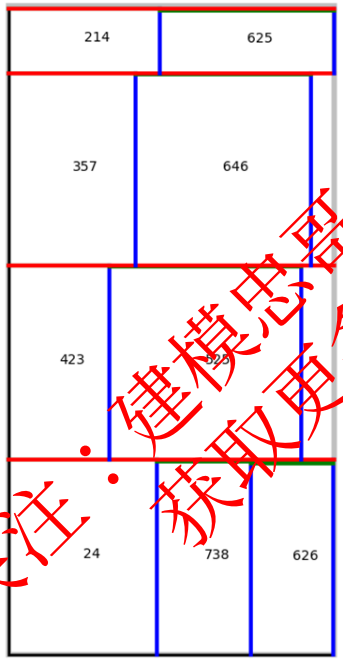

图 3-4 旋转优化效果图

图 3-4 展示了单块板材上的排样分布，其中产品项 214、产品项 625 在旋转后排样，组成新的 Stack，填补了板材最上方的剩余，如不考虑旋转，产品项 214、产品项 625 将在新的板材中进行排样，该板材将出现较多剩余。

3.4.3 产品价值修正策略

在初次排样时，我们按照产品项 i 的长度对其排序，得到 Item 序列 A，以此为基础进行排样。这会导致算法在实现时，总是从当前剩余 Item 中长度最大或宽度最大的 Item 开始搜索，优先选择长度或宽度较大的 Item 进行排样。当板材中剩余容量较小时，该算法耗时较长，需要进行多次搜索才能找到长度和宽度满足要求的 Item，此时长度较大的产品项无法及时排样，可能会造成组合困难，最终占用新的板材。为了降低排样过程中的搜索次数，更快找到符合要求的 Item，我们引入“产品项价值 v”，基于价值修正对产品项的顺序进行调整，在每次排样结束后更新产品项价值及排序，重新进行排样，如此迭代直到板材用量不再减少。主要分为 3 步：

（1）设定初始价值

对产品项 i 赋予初始价值。在首次排样时，我们基于产品项 i 的长度对其排序，因此，产品项 i 的初始价值 v_0 表示为：

$$v_{i0} = l_i,\ i = 1,\ldots,n \tag{3-20}$$

（2）确定价值修正算子

根据首次排样获取的板材利用率对产品项 i 的价值进行修正。板材利用率较低，说明排样在该板材上的产品项较难与其他产品组合，无法通过组合形成更优的排样方案。价值修正的目的是人为调整较难组合的产品项价值，通过调整其在产品序列中的优先级，对排样方案进行优化。产品项 i 的价值修正公式如下：

$$v_{iT} \leftarrow g_1 v_{iT-1} + g_2 l_i w_i^{\tau} / u_T \tag{3-21}$$

式中，产品项 i 的新价值由旧价值和修正价值组成。其中 T 表示第 T 次迭代，u_T 为第 T 次排样的板材利用率：

$$u_T = \frac{\sum_{i=1}^{n} l_i w_i}{mLW} \tag{3-22}$$

g_1、g_2 分别为旧价值权重和修正价值权重：

$$\begin{cases} g_2 = \upsilon \\ g_1 = 1 - g_2 \end{cases} \tag{3-23}$$

式中，τ, υ 是用于确定权重的经验参数；

由式（3-21）可知，修正价值与产品项长度正相关，产品项长度越大，修正后价值越大，排样时优先级越高；修正价值与材料利用率负相关，材料利用率越低，修正价值越大，该产品项在后续排样中优先级越高。经过多次修正，材料利用率低的排样方式包含的产品项价值有较大增加，在排样中被优先选择的机会变大。

（3）迭代生成最优排样方案

完成产品项价值修正后，根据修正后的价值对产品项重新排序，然后排样。每一次排样后选择板材用量最少的方案更新当前最优排样方案，多次迭代，直到满足设定的迭代次数。

3.5 求解结果与分析

经过多次迭代，本题最终输出的板材用量及板材利用率如表 3-2 所示：

表 3-2 数据集 A 板材用量、板材利用率、运行时间

数据集 A	**板材用量**	**板材利用率**	算法运行时间
dataA1	**89**	**93.87%**	0.24s
dataA2	**90**	**92.08%**	0.22s
dataA3	**89**	**94.08%**	0.24s
dataA4	**87**	**94.08%**	0.28s

由表 3-1 可知，基于不同数据集获得的最终排样方案中，板材利用率均大于 92%，数据集 A3、A4 获得的板材利用率最高，为 94.08%，这一结果表明，当采用本题建立的模型及算法进行排样设计时，板材浪费可以控制在 10%以内，从而实现降低生产成本及减少资源消耗的目标。

对问题一中所有生成的排样方案的板材利用率绘图如下：

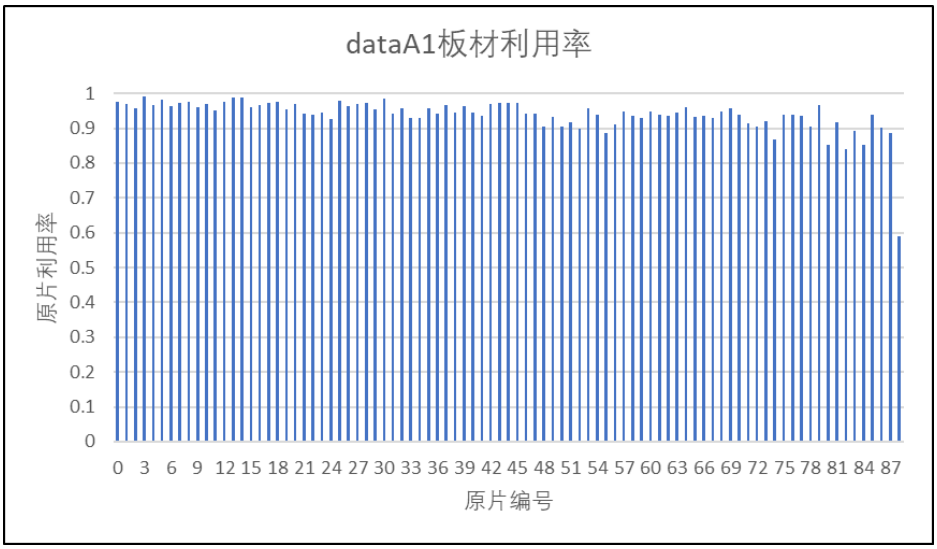

图 3-5 dataA1 板材利用率条形图

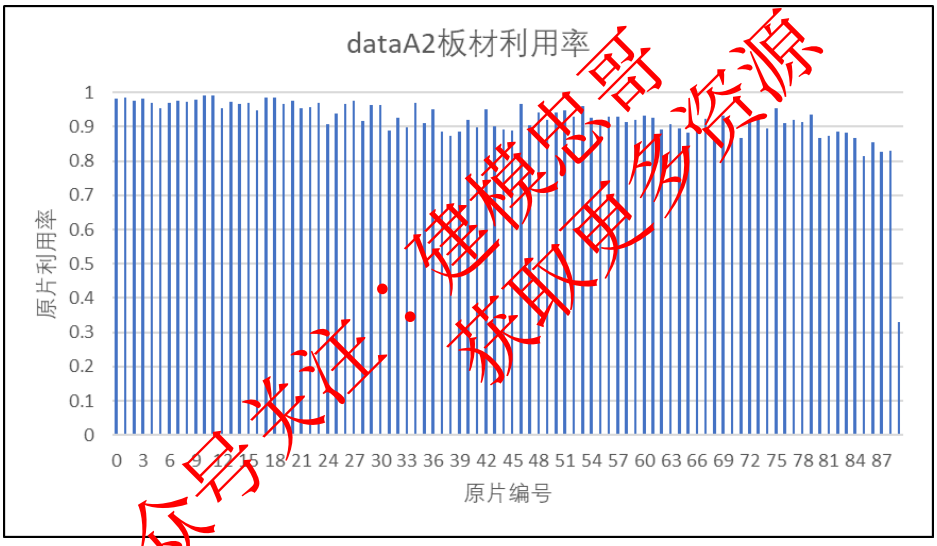

图 3-6 dataA2 板材利用率条形图

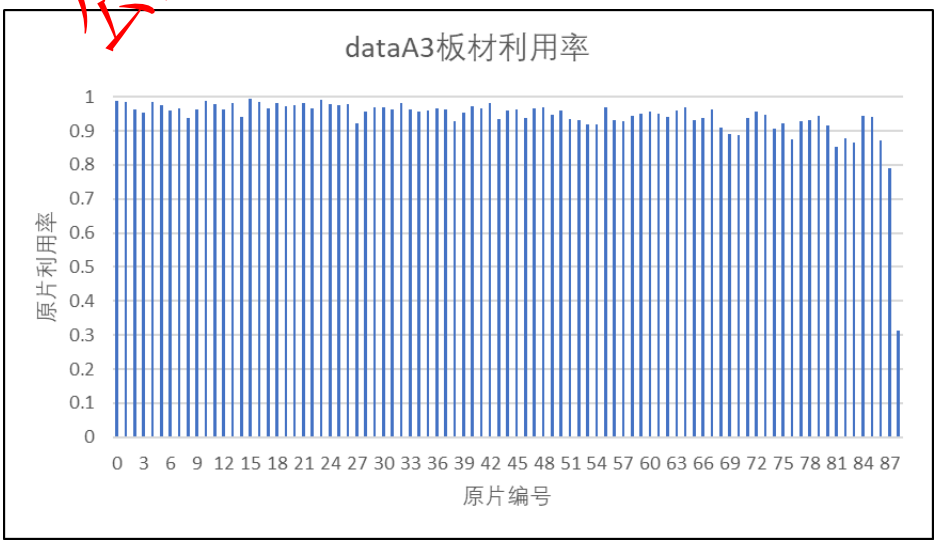

图 3-7 dataA3 板材利用率条形图

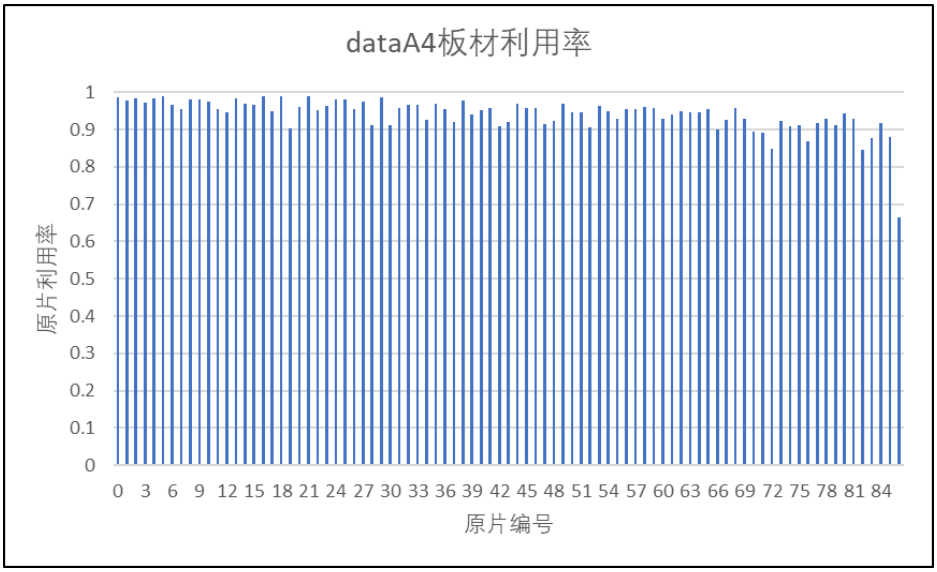

图 3-8 dataA4 板材利用率条形图

图 3-5 至 3-8 中，绝大部分板材利用率在 90%以上，个别利用率较低的板材可能是排样过程钟最后生成的，没有剩余产品项需要继续排样。

通过分析算法的流程结构以及算法的运行时间，得到算法的时间复杂度为$O(n^3)$。

除板材用量、板材利用率外，本题还通过建立坐标系对排样方案进行详细刻画，在结果中输出排样方案的详细坐标，并绘制排样方案效果图。

表 3-3　dataA1 求解输出的部分排样坐标

原片材质	原片序号	产品 id	产品 x 坐标	产品 y 坐标	产品 x 方向长度	产品 y 方向长度
ZQB-0218S	16	30524	0	0	2101	1151
ZQB-0218S	7	30909	0	0	2341	927
ZQB-0218S	2	30968	320	0	2396	847
ZQB-0218S	28	30538	0	0	2001	942
ZQB-0218S	11	30453	0	0	2281	736
ZQB-0218S	8	30882	0	0	2306	720
ZQB-0218S	37	30984	0	0	1701.5	960
ZQB-0218S	29	30573	0	0	1951	810

完整求解结果见对应数据集附件 cut_program.csv。

DataA1 所有板材排样方案如图 3-9 所示，对其中部分板材的排样图放大展示，如图 3-10。

图 3-10 中，黑色框线表示板材边框，每块板材上方的数字代表板材编号，板材内的数字代表该板材经过三阶段切割最终生成的 Item。红线为第一阶段的切割，生成多个 strip；蓝线为第二阶段的切割，生产多个 stack；绿线表示第三阶段的切割，生产多个 item，未标明数字的灰色封闭矩形为切割结束后的板材余料。

图 3-10 中多数 Item 的排样方向与板材方向一致，即板材长边与 Item 长边平行，这是因为在采用贪心策略进行排样时，每次搜索都先考虑长度条件是否满足；图中部分 Item 的排样方向与板材方向垂直，这是在采用产品项旋转策略的排样结果，如不考虑旋转，板材 73 中的 Item532、板材 76 中的 192 将无法在当前板材中进行排样，需要新建板材才能继续排样，由此可见，当 Item 可旋转时，板材利用率将会提高。

完整求解结果见附件。

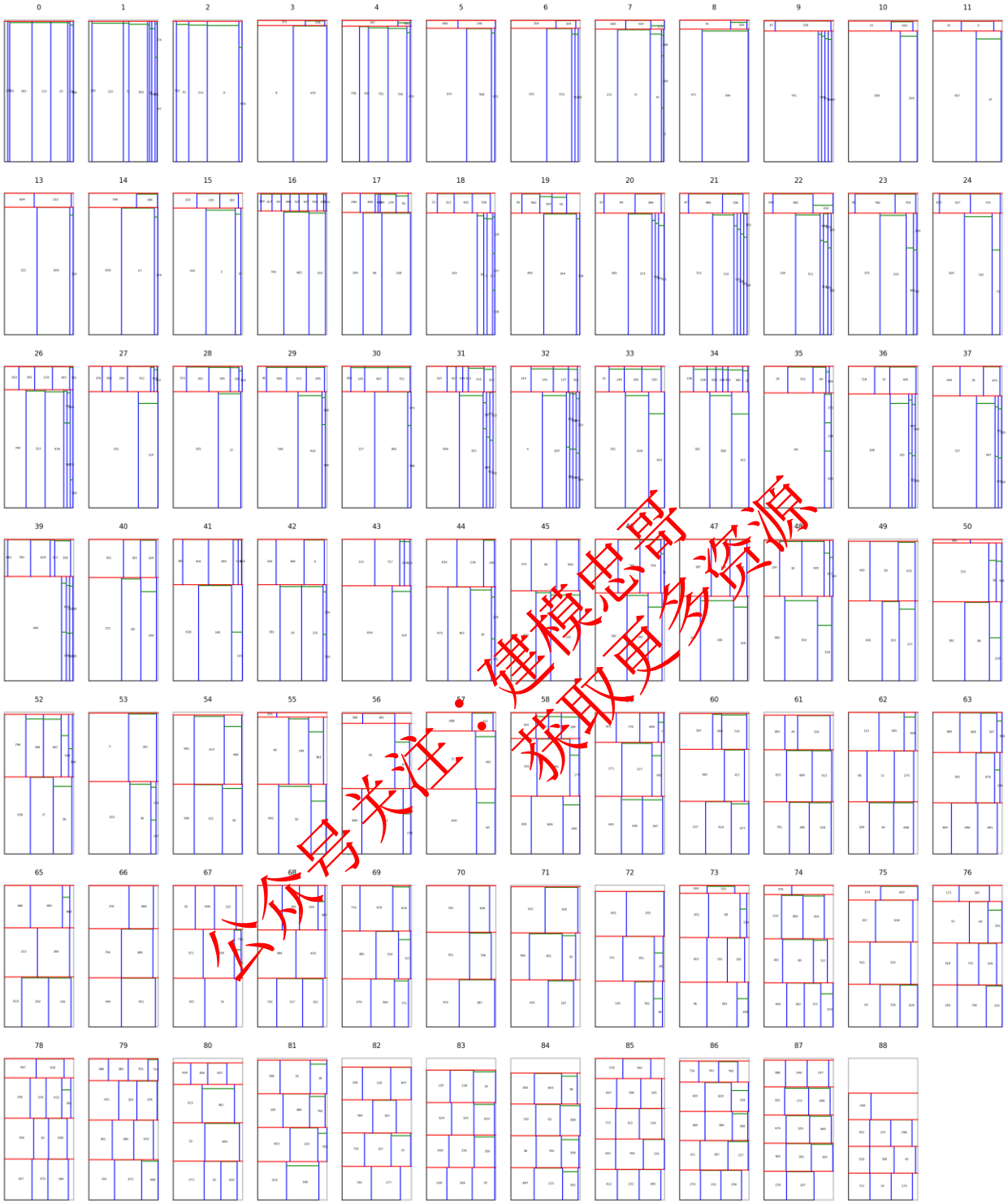

图 3-9 DataA1 所有板材排样方案效果图

图 3-10 问题一典型排样方案效果示意图

四、 问题二的模型建立与求解

4.1 问题分析

问题二在问题一的基础上引入了组批的概念，与问题一不同的是，问题二考虑了每个批次下 item 材质、总数、总面积、订单号的额外因素，而问题一中的约束条件在问题二中也同样需要考虑，优化的目标函数和问题一相同，即板材用量尽可能少，据此建立整数规划模型进行分析求解，提高模型的可解释性。

问题二相比问题一，核心和难点在解决组批问题，整体上首先需要将 item 根据订单号进行分组，在依据组批算法将每个订单组合成订单组批，之后得到每个批次的所有订单的 item，并对此按照材质进行分组，将每一个分组后的结果进行独立排样，问题二的排样优

化模型整体思路如图 4-1 所示：

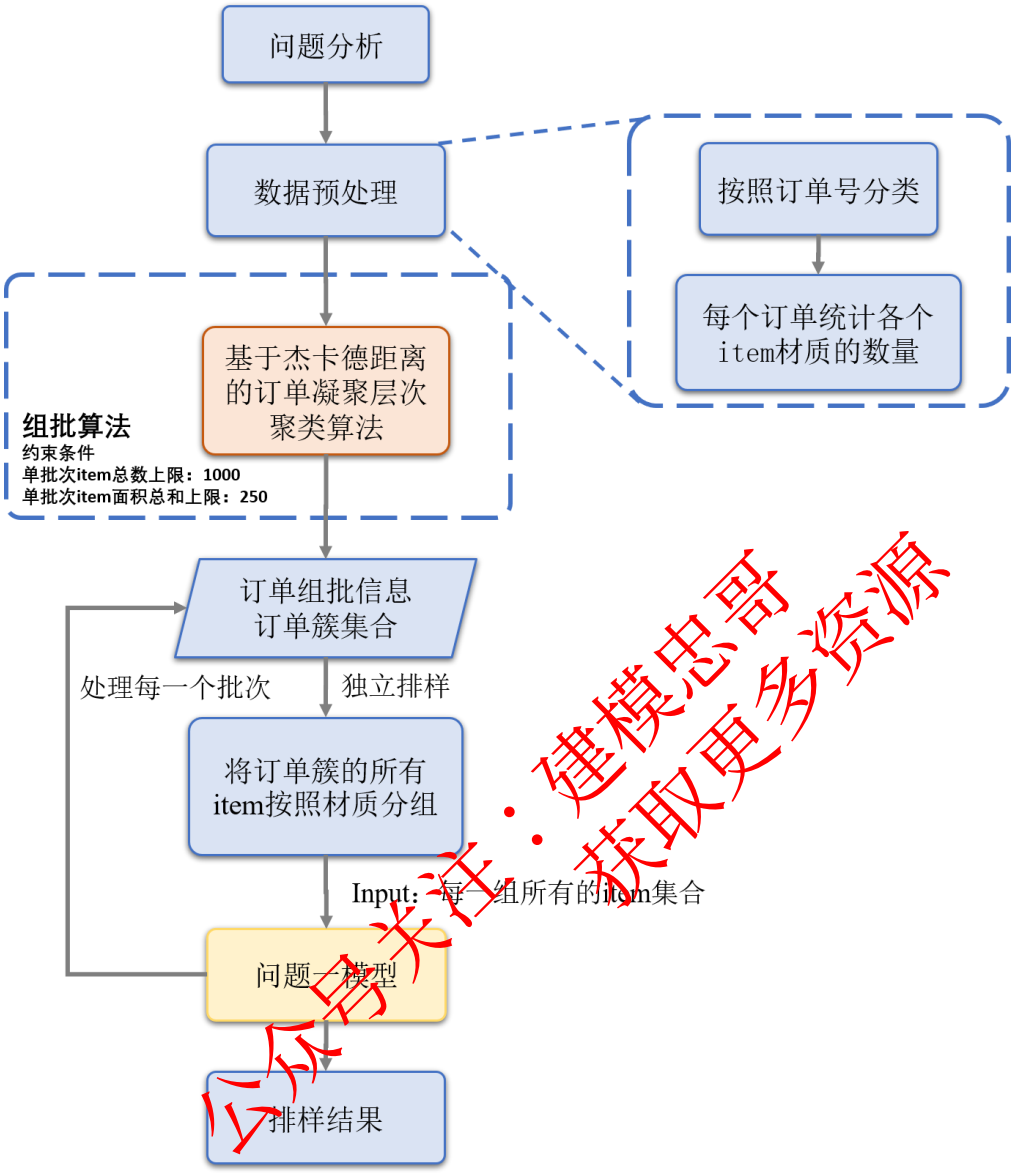

图 4-1 问题二分析思路图

在组批算法中，模型首先对每个订单所有的 item 材质进行统计，并以此为特征值，输入基于杰卡德距离的订单凝聚层次聚类算法，聚类过程需满足约束算法，并尽可能多的订单聚成一类，之后得到聚类后的订单组批，该聚类算法会将相似型号材质的订单尽量放在同一批次，以减少板材的用量。

4.2 数据预处理

读取文件信息，将所有 item 按照订单号进行分类，统计每个订单中的 item 总数和 item 总面积，并对每个订单中出现的所有材质类型进行特征编码，将材质作为特征名，统计每一个订单中该材质 item 的数量作为特征值，没有则为 0。

4.3 模型建立

问题二的优化目标依然是板材用量尽可能少，但需进一步考虑生产约束，即每个批次的产品项总数、产品项总面积的均有限制。我们在问题一的基础上，添加决策变量和生产约束条件，建立整数线性规划模型。

4.3.1 决策变量

在问题一的基础上，我们添加决策变量$\eta_{m,l}$、$\omega_{u,v}$用以区分不同订单和批次：

（1）当且仅当 Bin m 包含在批次 Batch u 中，$\eta_{y,m}$取 1.

$$\eta_{y,m} \in \{0,1\},\ y=1,\ldots,n,\ m=y,\ldots,n \tag{3-24}$$

（2）当且仅当订单 Order x 放在批次 Batch y 中，$\omega_{y,x}$取 1.

$$\omega_{y,x} \in \{0,1\},\ y=1,\ldots,n,\ x=1,\ldots,f \tag{3-25}$$

4.3.2 目标函数

本题的优化目标依然是板材用料尽可能少：

$$Z=\min\sum_{i=1}^{n}\gamma_{m,m} \tag{3-26}$$

4.3.3 约束条件

除问题一排样过程中长度、宽度的约束，本题进一步考虑每批生产中产品项总数、总面积的限制，在模型中添加如下约束：

（1）每块 Bin m 仅包含在 1 个批次 Batch y 中：

$$\sum_{u=1}^{m}\eta_{y,m}=\gamma_{m,m},\quad \forall m=1,\ldots,n \tag{3-27}$$

（2）每个批次 Batch y 中产品项的数量之和不超过 N.

$$\sum_{m=y}^{n}\sum_{k=m}^{n}\sum_{j=k}^{n}\sum_{i=j}^{n}\eta_{y,m}\gamma_{m,k}\beta_{k,j}\alpha_{j,i}\le N,\ \forall y=1,\ldots,n \tag{3-28}$$

（3）每个批次 Batch y 中产品项的面积之和不超过 S：

$$\sum_{m=y}^{n}\sum_{k=m}^{n}\sum_{j=k}^{n}\sum_{i=j}^{n}l_i w_i\eta_{y,m}\gamma_{m,k}\beta_{k,j}\alpha_{j,i}\le S,\ \forall y=1,\ldots,n \tag{3-29}$$

（4）每块 Bin m 上放置的产品 i 的材质相同：

$$\begin{aligned}&\alpha_{j,i}=0,\ \forall j=1,\ldots,n-1,\ \forall i>j\,|\,t_i\neq t_j\\&\beta_{k,j}=0,\ \forall k=1,\ldots,n-1,\ \forall j>k\,|\,t_j\neq t_k\\&\gamma_{m,k}=0,\ \forall m=1,\ldots,n-1,\ \forall k>m\,|\,t_k\neq t_m\end{aligned} \tag{3-30}$$

（5）每一个订单 Order x 只放置在一个批次 Batch y 中：

$$\begin{aligned}&\sum_{y=1}^{n}\sum_{x=1}^{z}\omega_{yx}=z\\&\sum_{y=1}^{m}\omega_{uv}=1\ ,\ \forall v=1,\ldots,z\end{aligned} \tag{3-31}$$

（6）每个产品的订单属性为所在批 Batch u 的订单属性：

$$\begin{aligned}&\lambda_{ym}\gamma_{mk}\beta_{kj}\alpha_{ji}=\omega_{yx}\ ,x=t_k,\\&\forall y=1,\ldots n,\forall m=y,\ldots,n,\forall k=m,\ldots,n,\forall j=k,\ldots,n,\forall i=j,\ldots,n\end{aligned} \tag{3-32}$$

综上，问题二建立的整数规划模型如下：

$$Z=\min\sum_{i=1}^{n}\gamma_{m,m}$$

$$s.t.\begin{cases}
\sum_{j=1}^{i}\alpha_{j,i}=1,\forall i=1,\ldots,n\\
\sum_{i=j+1}^{n}\alpha_{j,i}\leqslant(n-j)\alpha_{j,j},\ \ \forall j=1,\ldots n-1,\\
\alpha_{j,i}=0,\ \ \forall\ w_i\neq w_j\cup l_i+l_j>L,\ \text{其中}i>j,\ \ j=1,\ldots,n-1\\
\sum_{k=1}^{j}\beta_{k,j}=\alpha_{j,j},\quad \forall j=1,\ldots,n\\
\sum_{i=j}^{n}l_i\alpha_{j.i}<\sum_{i=k}^{n}l_i\alpha_{k,i}+(L+1)(1-\beta_{k,j}),\ \ \forall k=2,\ldots,n,\ \ \forall j=1,\ldots,k-1,\\
\sum_{i=j}^{n}l_i\alpha_{j.i}\leqslant\sum_{i=k}^{n}l_i\alpha_{k,i}+L(1-\beta_{k,j}),\ \ \forall k=1,\ldots,n-1,\ \ \forall j=k+1,\ldots,n,\\
\sum_{j=k}^{n}w_j\beta_{k,j}\leqslant W\beta_{k,k},\ \ \forall k=1,\ldots,n-1\\
\sum_{m=1}^{k}\gamma_{m,k}=\beta_{k,k},\quad \forall k=1,\ldots,n\\
\sum_{i=m}^{n}l_i\gamma_{m,i}+\sum_{i=m+1}^{n}l_i\sum_{j=m}^{i-1}\delta_{m,i,j}\leqslant L\gamma_{m,m},\ \ \forall m=1,\ldots,n-1\\
\alpha_{j,i}+\gamma_{m,j}-1\leqslant\delta_{m,i,j}\leqslant(\alpha_{j,i}+\gamma_{m,j})/2,\\
\qquad\forall m=1,\ldots,n-1,\ \forall i=m+1,\ldots,n,\ j=m,\ldots,i-1\\
\sum_{k=m+1}^{n}\gamma_{m,k}\leqslant(n-1)\gamma_{m,m},\ \ \forall m=1,\ldots,n-1\\
\sum_{u=1}^{m}\eta_{y,m}=\gamma_{m,m},\quad \forall m=1,\ldots,n\\
\sum_{m=y}^{n}\sum_{k=m}^{n}\sum_{j=k}^{n}\sum_{i=j}^{n}\eta_{y,m}\gamma_{m,k}\beta_{k,j}\alpha_{j,i}\leqslant N,\ \ \forall y=1,\ldots,n\\
\sum_{m=y}^{n}\sum_{k=m}^{n}\sum_{j=k}^{n}\sum_{i=j}^{n}l_iw_i\eta_{y,m}\gamma_{m,k}\beta_{k,j}\alpha_{j,i}\leqslant S,\ \ \forall y=1,\ldots,n\\
\alpha_{j,i}=0,\ \ \forall j=1,\ldots,n-1,\ \forall i>j\,|\,t_i\neq t_j\\
\beta_{k,j}=0,\ \ \forall k=1,\ldots,n-1,\ \forall j>k\,|\,t_j\neq t_k\\
\gamma_{m,k}=0,\ \ \forall m=1,\ldots,n-1,\ \forall k>m\,|\,t_k\neq t_m\\
\sum_{y=1}^{n}\sum_{x=1}^{z}\omega_{yx}=z\\
\sum_{y=1}^{m}\omega_{uv}=1\ ,\ \forall v=1,\ldots,z\\
\lambda_{ym}\gamma_{mk}\beta_{kj}\alpha_{ji}=\omega_{yx}\ ,x=t_k,\\
\qquad\forall y=1,\ldots n,\forall m=y,\ldots,n,\forall k=m,\ldots,n,\forall j=k,\ldots,n,\forall i=j,\ldots,n
\end{cases}\tag{3-33}$$

4.4 模型求解

本题采用基于凝聚层次聚类的启发式算法求解订单组批及排样问题，算法主要流程是先对所有 item 按照订单号分组，在对所有订单进行组批聚类，最后将每一个组批的 item 按照材质分组输入模型一进行独立排样。

算法实现的核心步骤如下：

表 4-1 问题二算法实现核心步骤

基于凝聚层次聚类的启发式算法
(1) **订单特征选择：**由于每份订单当且仅当出现在一个批次中，因此需要对订单的所有信息进行特征处理，根据订单的特征来进行订单组批的构建。通过分析题意，每个订单中存在不同材质和数量的 item，又由于每个批次中的相同材质的 item 才能使用同一块板材原片进行排样，所以板材原片的用量和 item 的材质紧密相关，因此模型将订单存在的 item 材质和数量作为订单的重要特征，提高模型的可解释性。 (2) **基于杰卡德距离的凝聚层次聚类算法：**对订单特征进行选择后，需要对订单进行组批聚类，考虑到订单 item 总数和订单 item 总面积的限制，模型采用自底向上的的凝聚层次聚类，在聚类过程中采用杰卡德距离作为标准，将相似度越高的订单簇进行合并，并在合并过程中判断约束条件，从而提高模型的高效性和可行性。 (3) **订单组批优化策略：**因为一个订单中相同板材的 item 越多，对其进行排样后的利用率也就越大，基于上述贪心思想，模型在聚类过程中尽可能让每一个订单簇的总 item 数和总面积接近约束条件值，将此作为第一个贪心策略。而在组批中发现若订单组批只有少量某种材质的 item，则该订单组批的利用率将会被降低，因此模型将只有少量某种材质 item 的订单与其他有大量该材质 item 的订单进行组批聚类，作为模型的第二个贪心策略，优化了模型的排样效果。

4.4.1 算法主框架

本节首先介绍算法的主体框架，其中涉及到的策略和子算法在后续详细介绍。

算法 3 基于凝聚层次聚类的启发式算法

输入：所有的 item 序列

输出：所有 item 按照组批的排样结果

Step1：将所有 item 按照订单号进行分组

Step2：统计所有订单中存在的所有 item 的材质类型，将材质作为特征名，将订单中该材质的 item 数量作为特征值，没有则为 0。

Step3：根据杰卡德距离对订单进行订单凝聚层次聚类，具体步骤见 4.4.1

Step4：结束组批，开始对每个组批的所有订单 item 进行排样优化

Step5：遍历所有组批，对于每一个组批，获取当前组批所有订单的所有 item 信息。

Step6：对所有 item 按照材质进行分组。

Step7：对每一个材质分组的 item 序列作为第一问模型的输入，得到排样结果。

Step8：重复 Step5-Step7,直到得到所有组批 item 的排样结果

Step9：算法结束

问题二算法流程如图 4-2 所示：

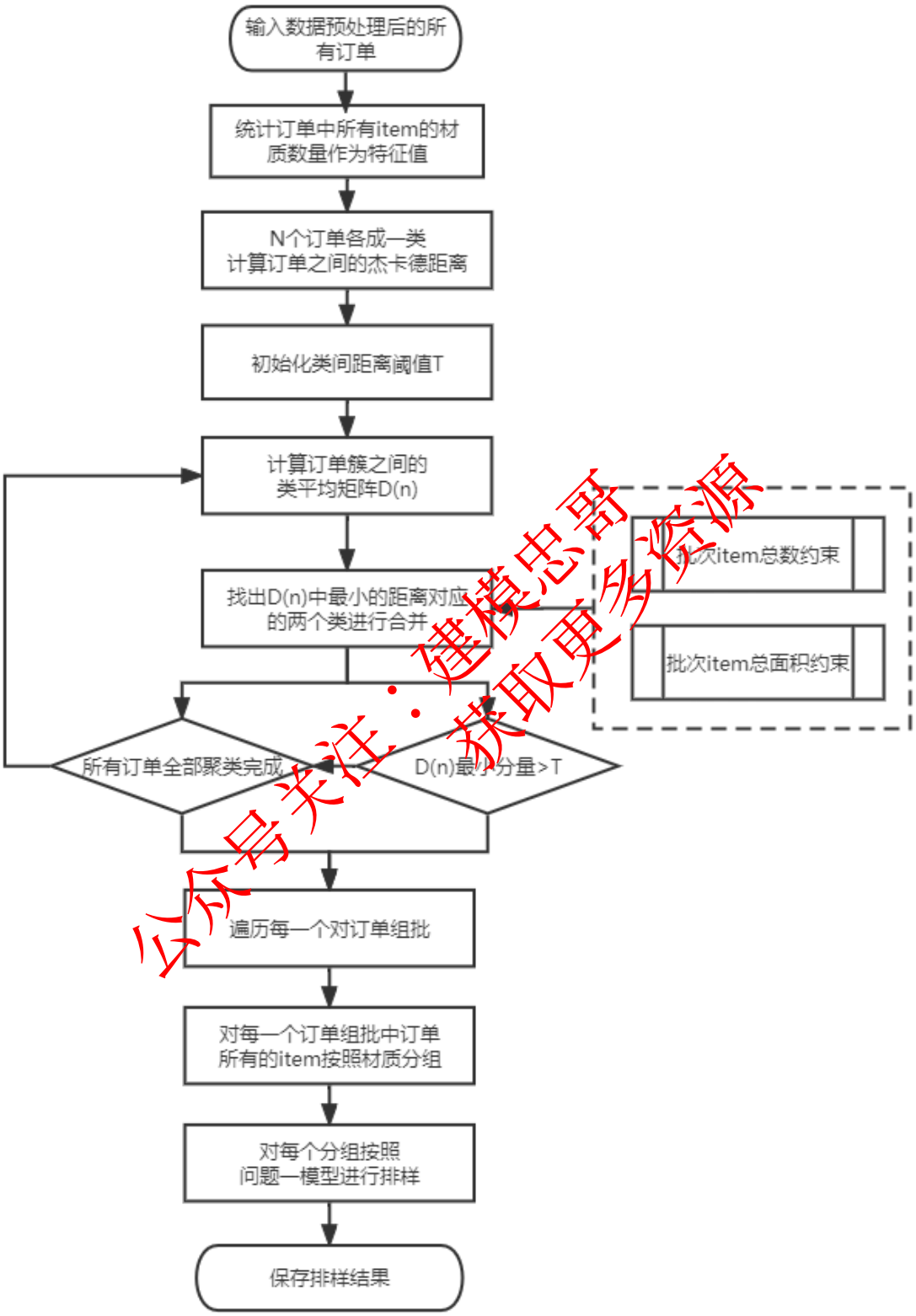

图 4-2 问题二算法流程图

4.4.2 订单组批求解策略

方形件下料的组批通常按照不同生产订单的相似性划分成多个批次，从而满足订单的个性化需求同时保持生产的高效性。根据题目要求，不考虑不同订单的交货期的影响，订

单组批考虑的主要因素包括订单中包含的不同材质和不同材质产品的数量。

我们使用杰卡德相似系数和杰卡德距离来衡量两个订单批次的相似度。

（1）杰卡德相似系数：两个批次 A 和 B 的材质种类交集在 A， B 的并集中所占的比例，称为两个订单集合的杰卡德相似系数，用符号 $J(A,B)$ 表示：

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|} \tag{3-34}$$

（2）杰卡德距离：用两个批次中不同材质与所有元素比例来衡量集合区分度。

$$J_\delta(A,B)=1-J(A,B)=\frac{|A\cup B|-|A\cap b|}{|A\cup B|} \tag{3-35}$$

我们使用类平均距离法来计算订单组批的类间距离。类平均距离法是计算两个簇之间所有距离的均值作为两个订单簇之间的距离。

订单簇之间距离计算公式：

$$D_{HK}=\sqrt{\frac{1}{n_H n_k}\sum_{\substack{i\in H\\ j\in K}} d_{ij}^2} \tag{3-36}$$

式中，d_{ij}^2 表示类任一样本和类任一样本之间的欧式距离平方。

若 K 类由 I 类和 J 类合并产生，则递推式为：

$$D_{HK}=\sqrt{\frac{n_I}{n_I+n_J}D_{HI}^2+\frac{n_J}{n_I+n_J}D_{HJ}^2} \tag{3-37}$$

基于生产约束的订单层次聚类算法的关键步骤如下：

Step1： N 个初始状态下的订单各自成一类，建立 N 个类，计算各类之间的距离得到一个 $N*N$ 维的距离矩阵。

Step2：计算各批次之间的类平均距离，找出其中距离最小得到对应的两个批次。

Step3：尝试将两个批次合并成一个新批次，判断新批次的产品总面积和总产品数是否超出约束。如果不满足约束，则返回 Step2，找出次小的对进行合并并检验；如果满足题目约束条件，则进行合并，删除旧的批次，建立一个新批次。

Step4：计算合并后批次之间的新距离。

Step5：跳至步骤二，重复计算及合并，直到所有批次合并后都不满足约束条件。

订单凝聚层次聚类过程如图 4-3 所示。

在聚类过程中，每次批次合并前先判断两个批次合并后是否满足题目的约束条件，产品总面积大小不超过 250，产品总数量不超过 1000。如果满足约束条件则合并，不满足则再寻找更小的批次间距离进行合并，直到不能合并为止。满足题目生产约束的订单凝聚层次聚类结果如图 4-4 所示（为部分样例）。

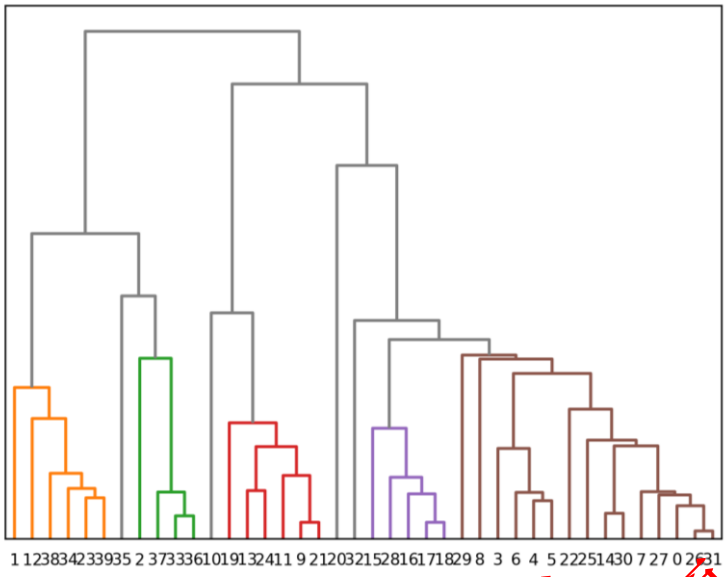

图 4-3 凝聚层次聚类图示

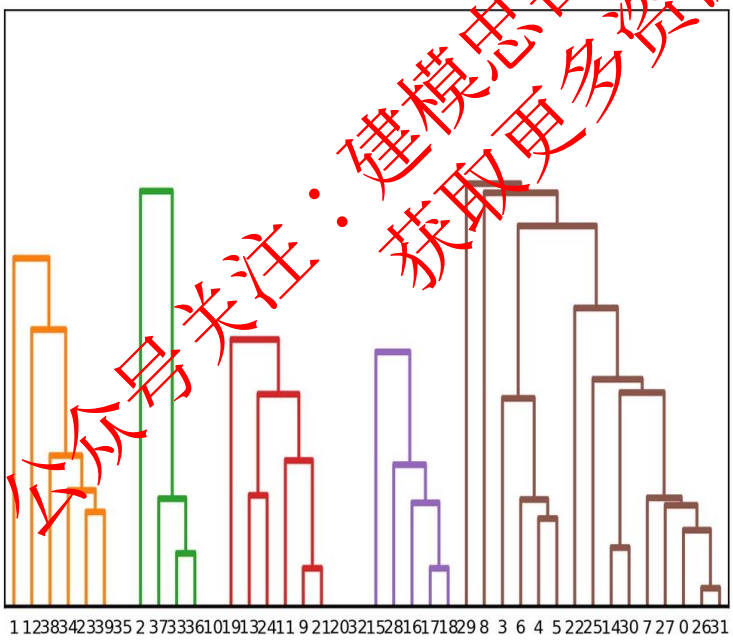

图 4-4 订单凝聚层次聚类结果图示

订单凝聚层次聚类部分结果如表 4-2 所示：

表 4-2 订单凝聚层次聚类部分结果

组批号	订单 id
0	order107，order168，order195，order199，order200，order278，order279，order436，order492，order517，order518，order519，order520，order522，order523，order525
1	order7，order24，order25，order28，order175，order176，order177，order178，order180，order182，order183，order265，order266，order267，order269，order397，order399，order403，order507
2	order6，order92，order93，order241，order257，order258，order263，order358，order359

3	order51，order52，order53，order54，order272，order273，order274，order275，order276，order429，order475
4	order1，order2，order60，order64，order374，order546
5	order34，order35，order39，order135，order138，order139，order140，order141，order174，order191，order193，order227，order228，order284，order322，order363，order395，order419，order430，order471
6	order44，order213，order256，order329，order330，order331，order333，order334，order335
7	order18，order27，order161，order372，order506，order508，order509
8	order211，order212，order280，order281，order282，order283
9	order173，order222，order311，order312，order317，order319，order320，order386

4.5 求解结果

经过订单组批及分批次排样优化，本题最终输出的板材用量及板材利用率如表 4-3 所示：

表 4-3 数据集 B 板材用量，板材利用率，算法运行时间

数据集 B	**板材数量**	**板材利用率**	运行时间
dataB1	**3863**	**77.10%**	18.73s
dataB2	**2550**	**75.56%**	7.76s
dataB3	**2589**	**74.69%**	7.70s
dataB4	**2666**	**76.15%**	6.62s
dataB5	**4123**	**74.94%**	23.78s

由表 4-3 可知，基于不同数据集获得的最终排样方案中，板材利用率在 75%左右。对比问题一，在单个批次产品总数及产品总面积的约束下，板材利用率有所下降，但仍保持较高水平。当采用本文建立的模型及算法进行订单组批及排样优化时，板材余料可控制在 25%左右。

通过分析算法的流程，对比算法的运行时间，可知该算法的时间复杂度为$O(n^3)$。

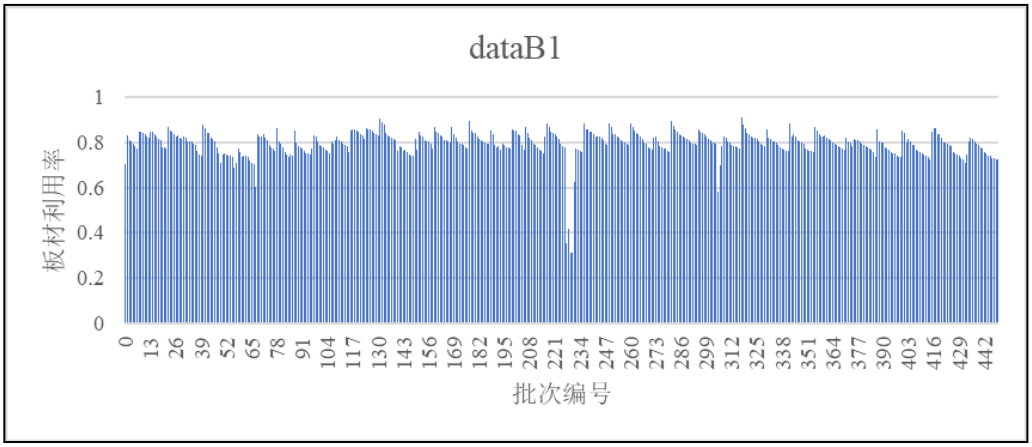

图 4-5 数据集 B1 组批排样后板材利用率

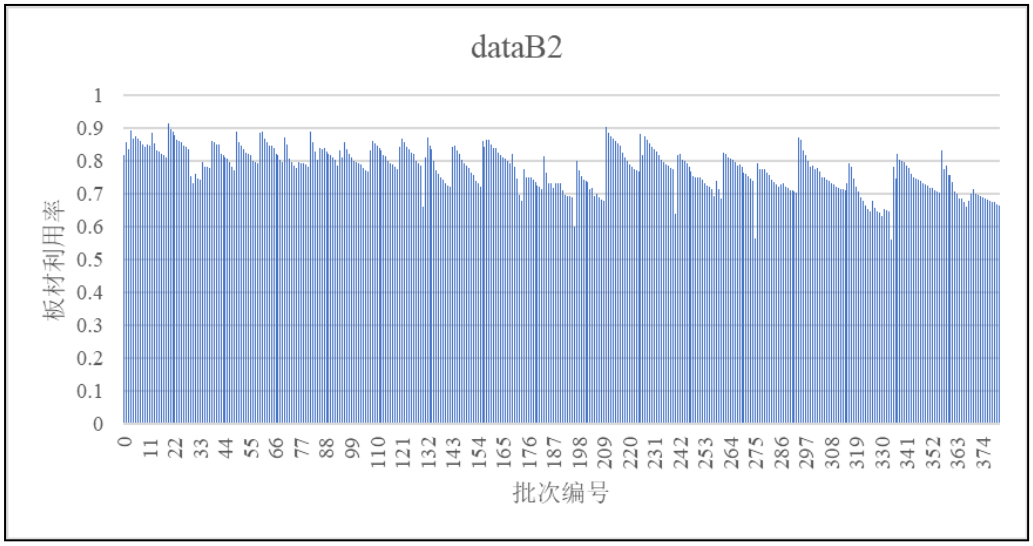

图 4-6 数据集 B2 组批排样后板材利用率

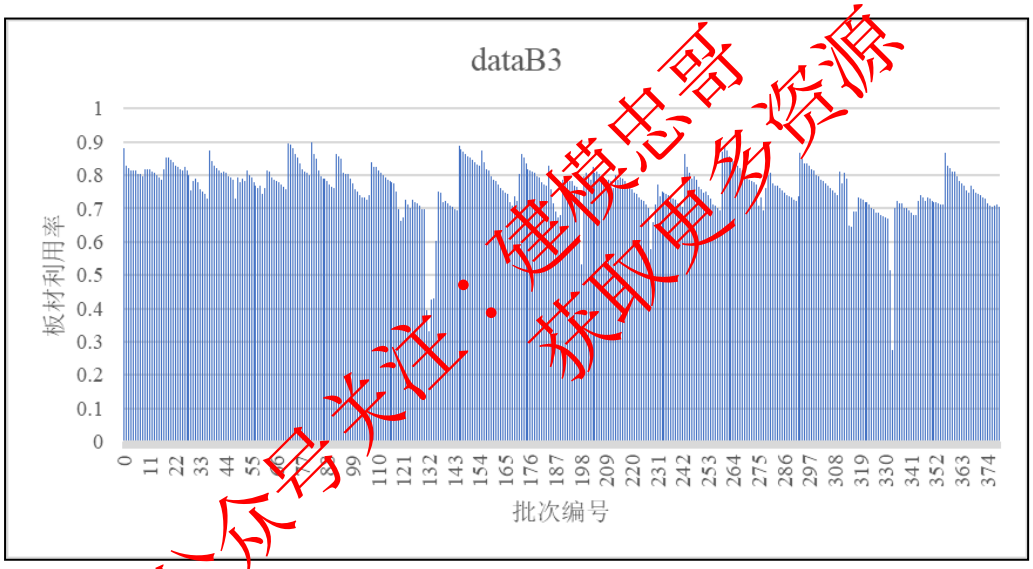

图 4-7 数据集 B3 组批排样后板材利用率

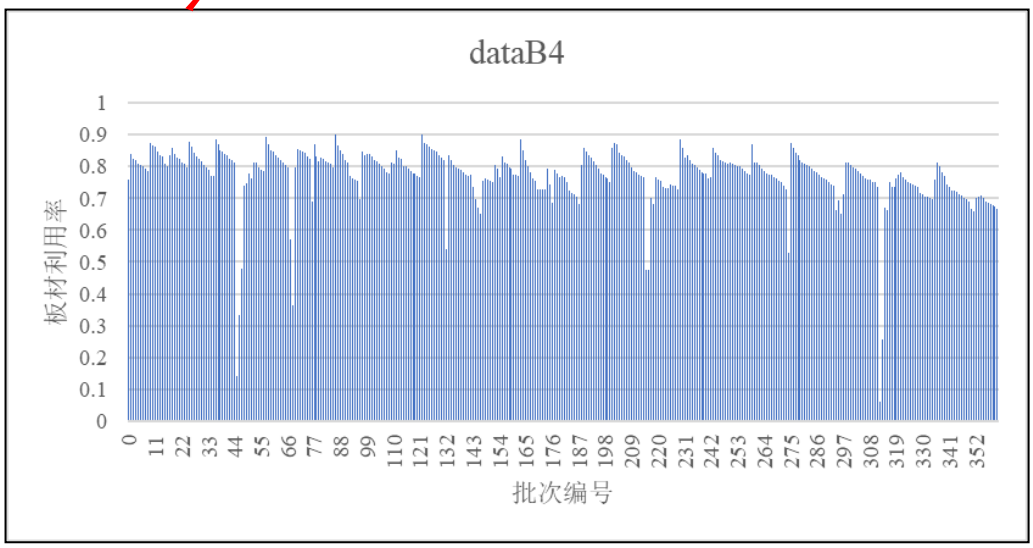

图 4-8 数据集 B4 组批排样后板材利用率

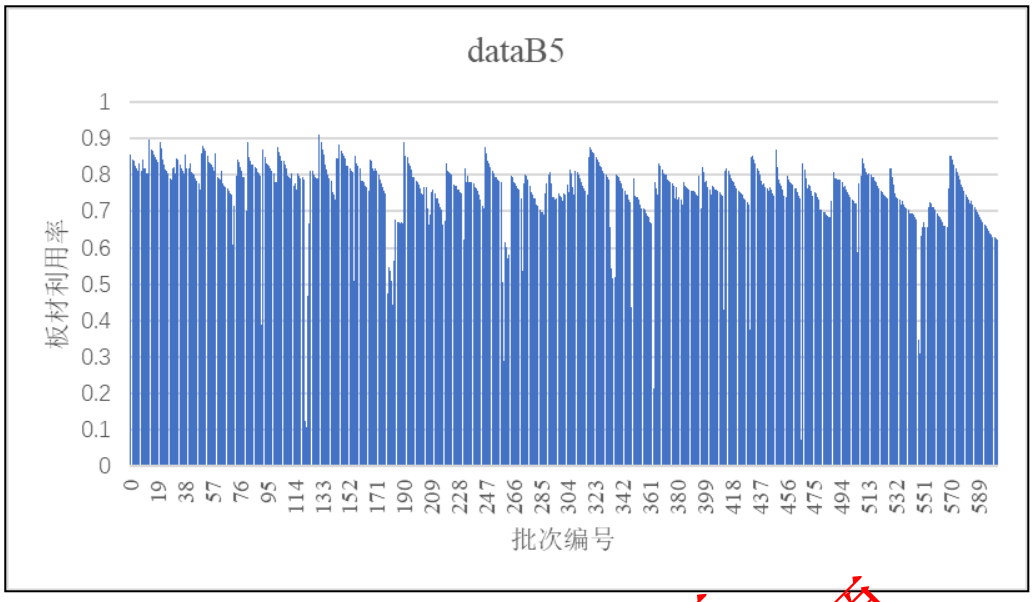

图 4-9 数据集 B5 组批排样后板材利用率

由图 4-5 至 4-9 可知，基于不同数据集获得的最终排样方案中，每个批次板材利用率在 75%附近波动，可见该模型对构建不同批次板材的利用率影响较小，具有良好的泛化能力。

除板材用量、板材利用率外，本题还通过建立坐标系对排样方案进行详细刻画，在结果中输出每块板材中每个 Item 的详细坐标，并绘制排样方案效果图。

表 4-4 部分板材排样坐标

批次序号	原片材质	原片序号	产品 id	产品 x 坐标	产品 y 坐标	产品 x 方向长度	产品 y 方向长度
0	QKQ-0218SD	0	4091	0	0	58	2418
0	QKQ-0218SD	1	4388	0	0	58	2418
0	QKQ-0218SD	0	5961	58	0	58	2408
0	QKQ-0218SD	0	4124	116	0	58	2408
0	QKQ-0218SD	0	6335	174	0	58	2400
0	QKQ-0218SD	0	3640	232	0	148	2398
0	QKQ-0218SD	0	7394	380	0	58	2398
0	QKQ-0218SD	0	4360	438	0	58	2388
0	QKQ-0218SD	0	4589	496	0	58	2388

完整求解结果见对应数据集附件 sum_order.csv。

图 4-10 展示了部分批次中部分材质的排样方式。板材上方编号依次表示批次号、材质标号、板材序号，如第二行第二例的示意图表示第 1 批次中材质 YSH-0218SD 第 10 块版的排样方式。

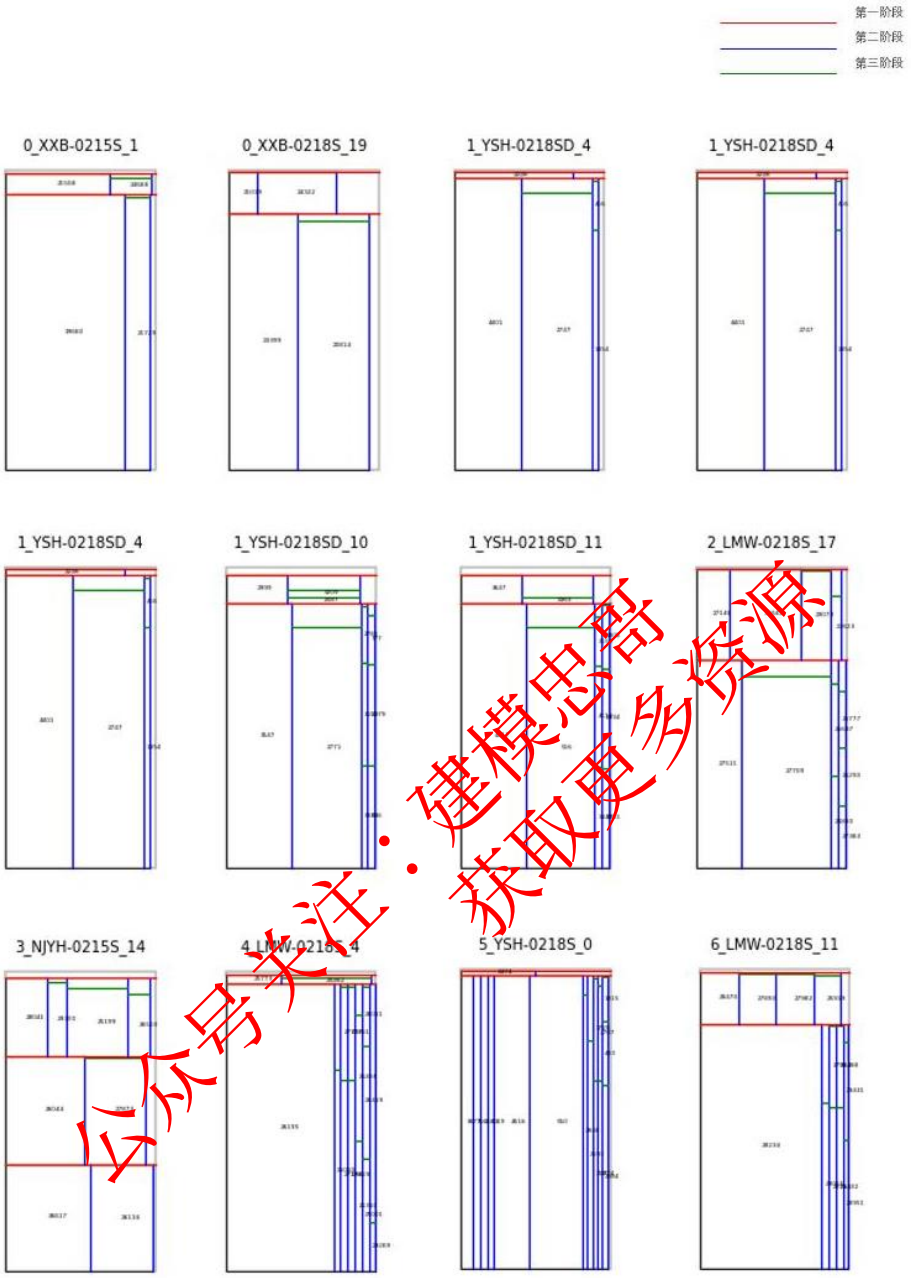

图 4-10 问题二典型批次排样图

图 4-11 展示了批次 0 中所有材质的所有排样方案。

图 **4-11**　问题二单个批次排样图

五、 模型及算法评价

5.1 算法的有效性和复杂度分析

问题一采用基于规则的启发式算法寻找满足约束条件与优化目标的解，求解算法中融入了贪心算法的思想，首先对每个 item 按照长宽进行排序，据此确定每个 stripe 的高度，遍历每个 item 进行排样堆叠，尽可能占满每一个 stripe 的空间，而每个 stripe 又尽可能占满每个板材的空间，为了让 item 的取值灵活易于调整，模型又考虑了 item 的旋转问题，这些排样在绝大部分情况下是局部最优解。因此在此基础上算法又为每个 item 添加了价值属性，在每一轮排样结束后，会重新对 item 进行价值修正，并按 item 的价值从大到小排序，之后对重新排序过后的 item 重新进行排样，迭代 N 次，这使得算法在一定程度上摆脱了贪心算法的局部性，能够得到一个相对于全局的较优解，该模型也是后续组批排样任务的基本单位，起到承前启后的作用，运行的时间主要体现在迭代排样中，空间复杂度为$O(nd)$，时间复杂度约为$O(n^3)$。

问题二的组批算法采用了基于杰卡德距离的订单凝聚层次聚类算法，该算法在全局的订单空间中搜索订单距离和规则相似度较高的订单簇，并进行合并，在算法中是逐步收敛的，能够得到一个全局的较优组批，之后对每个组批进行单独排样，对每个组批的所有订单 item 按照材质进行分组，对每个分组输入问题一的模型中进行求解，该算法的空间复杂度也在聚类中体现，凝聚层次聚类算法的空间复杂度为$O(nd+n^2)$，时间复杂度约为$O(n^3)$。

5.2 优点分析

（1）本文针对下料排样问题建立了单目标线性优化模型，能够有效解决下料三阶段排样问题。

（2）本文结合问题特征针对各问题分别提出了有效的启发式规则，可以在短时间内得到高质量的结果，通过将物品组合为 stack、strip、bin 等单位来简化问题的求解过程。

5.3 缺点分析

（1）建立的整数规划模型计算量大，在大数据量下难以进行快速求解。

（2）设计的启发式算法并不能找到全局最优解，只能找到局部最优解。

5.4 模型推广

（1）本文从组批下料问题出发，综合考虑组批的约束条件，使模型有一定的应用价值。

（2）三阶段排样模型的建立对于生活中其他类似规划问题也起着一定的借鉴作用。

参考文献

[1]陈秋莲. 二维剪切下料问题的三阶段排样方案优化算法研究[D].华南理工大学,2016.

[2]Puchinger J, Raidl G R. Models and algorithms for three-stage two-dimensional bin packing [J] European Journal of Operational Research, 2007, 183(3): 1304-1327.

[3] 张浩. 面向板式产品定制生产的组批与排样协同优化方法[D]. 广东工业大学,2019.DOI:10.27029/d.cnki.ggdgu.2019.001470.

[4] 阎春平. 面向物料资源优化利用的产品设计系统与优化下料技术研究[D].重庆大学,2002.

[5]贾志欣.排样问题的研究现状与趋势[J].计算机辅助设计与图形学学报,2004(07):890-897.

附录

问题 1 代码	代码功能： 求解所有 item 的排样

```
import copy

import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as patches

datapath = "dataA4"
df = pd.read_csv(f'子问题 1-数据集 A/{datapath}.csv')

class Res_item:
    def __init__(self, item_id, item_material, item_num, item_length, item_width, item_order):
        self.item_id = item_id
        self.item_material = item_material
        self.item_num = item_num
        self.item_length = item_length
        self.item_width = item_width
        self.item_order = item_order
        self.item_area = item_length * item_width
        self.item_value = item_length * item_width

    def set_item_value(self, value):
        self.item_value = value

    def set_item_pos(self, mate_id, x, y):
        self.mate_id = mate_id
        self.x = x
        self.y = y


class Strip:
    def __init__(self, height, uheight, width, uwidth, y):
        self.height = height
        self.uheight = uheight
        self.width = width
        self.uwidth = uwidth
        self.y = y

    def set_strip_stack(self, stack):
        self.stack = stack
```

```
class Stack:
    def __init__(self, height, width):
        self.height = height
        self.width = width

    def set_stack_items(self, items):
        self.items = items


class Bin:
    def __init__(self, height, uheight, width, bin_id):
        self.height = height
        self.uheight = uheight
        self.width = width
        self.bin_id = bin_id

    def set_bin_strips(self, strips):
        self.strips = strips

    def set_bin_area(self, area):
        self.cover_area = area

all_area = 0
res_items = []
for index, row in df.iterrows():
    # 同一宽要小，长要大
    if row.item_length > row.item_width:
        res_item = Res_item(row.item_id, row.item_material, row.item_num, row.item_length,
row.item_width,
                            row.item_order)
    else:
        res_item = Res_item(row.item_id, row.item_material, row.item_num, row.item_width,
row.item_length,
                            row.item_order)
    all_area += row.item_width * row.item_length
    res_items.append(res_item)

# 计算程序运行时间
import time
time_start = time.time()  # 记录开始时间

plate_length = 2440
plate_width = 1220
```

```
    # 长由大到小排序
    # res_items = sorted(res_items, key = lambda item: item.item_length*item.item_width, reverse=True)
    res_items = sorted(res_items, key=lambda item: [item.item_length, item.item_width], reverse=True)


    def first_fit(res_items):

        is_visit = [False for _ in range(len(res_items))]
        cur_mate = 0
        # 切割好所有的 item
        solution = []
        while False in is_visit:
            # 新建原片 bin
            res_bin = []
            is_end = False
            uheight = plate_length
            uwidth = plate_width
            strip_y = 0
            while not is_end:
                strip = []
                min_height = 0
                is_end = True
                cur_y = strip_y
                cur_x = 0

                strip_height = 98921
                cur_strip_height = 98921
                for index, res_item in enumerate(res_items):
                    # 新建 strip
                    # 当前的 item 满足放入的要求
                    if res_item.item_length < uheight and res_item.item_width < uwidth and not is_visit[index] and res_item.item_length < strip_height:
                        is_end = False
                        # 放入该 item 更新信息
                        if len(strip) == 0:
                            res_item.set_item_pos(cur_mate, cur_x, cur_y)

                            res_stack = []
                            res_stack.append(res_item)
                            stack_res = Stack(    res_item.item_length , res_item.item_width)
                            stack_res.set_stack_items(res_stack)
```

```
                strip.append(stack_res)
                is_visit[index] = True

                min_height = uheight - res_item.item_length
                cur_x += res_item.item_width
                uwidth -= res_item.item_width
                strip_height = res_item.item_length
                cur_strip_height = strip_height
                continue

            res_stack = []
            res_stack.append(res_item)
            is_visit[index] = True
            res_item.set_item_pos(cur_mate, cur_x, cur_y)
            # 限制条件更新
            uheight -= res_item.item_length
            uwidth -= res_item.item_width
            cur_y = cur_y + res_item.item_length
            cur_strip_height -= res_item.item_length
            # 遍历其他 item，宽度相等则放入 stack
            for i in range(0, len(res_items)):
                if res_items[i].item_width == res_item.item_width and
res_items[i].item_length < uheight and not is_visit[i] and res_items[i].item_length < cur_strip_height:
                    # 当前 item 放入 stack
                    res_stack.append(res_items[i])
                    res_items[i].set_item_pos(cur_mate, cur_x, cur_y)
                    is_visit[i] = True
                    cur_y += res_items[i].item_length
                    uheight -= res_items[i].item_length
                    cur_strip_height -= res_items[i].item_length

            # res_items_width = sorted(res_items, key=lambda item: item.item_width,
reverse=True)
            # 遍历其他 item，长度相等则放入 stack

            width_id = sorted(range(len(res_items)), key=lambda k:
res_items[k].item_width, reverse=True)
            for i in width_id:
                if not is_visit[i] and \
                        res_items[i].item_length == res_item.item_width and \
                        res_items[i].item_width < uheight and \
                        res_items[i].item_width < cur_strip_height:
                    res_items[i].item_length, res_items[i].item_width =
res_items[i].item_width, \
```

```
res_items[

i].item_length
                                    res_stack.append(res_items[i])
                                    res_items[i].set_item_pos(cur_mate, cur_x, cur_y)
                                    is_visit[i] = True
                                    cur_y += res_items[i].item_length
                                    uheight -= res_items[i].item_length
                                    cur_strip_height -= res_items[i].item_length


                            if uheight < min_height or min_height == 0:
                                min_height = uheight

                            # 当前 stack 遍历完成，加入 stripe
                            stack_res = Stack(cur_y - strip_y, res_item.item_width)
                            stack_res.set_stack_items(res_stack)
                            strip.append(stack_res)
                            # 更新条件
                            cur_strip_height = strip_height
                            uheight = plate_length - strip_y
                            cur_x += res_item.item_width
                            cur_y = strip_y

                    width_id = sorted(range(len(res_items)), key=lambda k: res_items[k].item_width,
reverse=True)
                    for index in width_id:
                        if res_items[index].item_width < uheight and \
                                res_items[index].item_length < uwidth and \
                                not is_visit[index] and \
                                res_items[index].item_width < strip_height:
                            is_end = False
                            if len(strip) == 0:
                                res_items[index].set_item_pos(cur_mate, cur_x, cur_y)

                                res_stack = []
                                res_stack.append(res_items[index])  # 转换宽高
                                res_items[index].item_length,      res_items[index].item_width      =
res_items[index].item_width, \

res_items[index].item_length
                                stack_res              =              Stack(res_items[index].item_length,
res_items[index].item_width)
```

```
                    stack_res.set_stack_items(res_stack)

                    strip.append(stack_res)
                    is_visit[index] = True

                    min_height = uheight - res_items[index].item_length
                    cur_x += res_items[index].item_width
                    uwidth -= res_items[index].item_width
                    strip_height = res_items[index].item_length
                    cur_strip_height = strip_height
                    continue
                res_stack = []
                res_items[index].item_length, res_items[index].item_width = res_items[index].item_width, res_items[
                    index].item_length
                res_stack.append(res_items[index])
                is_visit[index] = True
                res_items[index].set_item_pos(cur_mate, cur_x, cur_y)

                uheight -= res_items[index].item_length
                uwidth -= res_items[index].item_width
                cur_y = cur_y + res_items[index].item_length
                cur_strip_height -= res_items[index].item_length
                for i in width_id:
                    if not is_visit[i] and \
                            res_items[i].item_length == res_items[index].item_width and \
                            res_items[i].item_width < uheight and \
                            res_items[i].item_width < cur_strip_height:
                        res_items[i].item_length, res_items[i].item_width = res_items[i].item_width, \

res_items[

i].item_length
                        res_stack.append(res_items[i])
                        res_items[i].set_item_pos(cur_mate, cur_x, cur_y)
                        is_visit[i] = True
                        cur_y += res_items[i].item_length
                        uheight -= res_items[i].item_length
                        cur_strip_height -= res_items[i].item_length
                if uheight < min_height or min_height == 0:
                    min_height = uheight
                stack_res = Stack(cur_y - strip_y, res_items[index].item_width)
```

```
                        stack_res.set_stack_items(res_stack)
                        strip.append(stack_res)

                        cur_strip_height = strip_height
                        uheight = plate_length - strip_y
                        cur_x += res_items[index].item_width
                        cur_y = strip_y

                if len(strip) != 0:
                    res_strip = Strip( plate_length-strip_y-min_height , min_height, cur_x, plate_width-cur_x, strip_y)
                    res_strip.set_strip_stack(strip)
                    res_bin.append(res_strip)
                    strip_y = plate_length - min_height
                    uheight = min_height
                    uwidth = plate_width
            bin_solution = Bin(plate_length, uheight ,plate_width, cur_mate)
            bin_solution.set_bin_strips(res_bin)
            solution.append(bin_solution)
            cur_mate += 1
        print(cur_mate)
        return solution


max_G = 1
best_solution = None
while max_G != 0:
    max_G -= 1
    solution = first_fit(res_items)

    if best_solution == None or len(best_solution) > len(solution):
        best_solution = copy.deepcopy(solution)
    res_items = sorted(res_items, key=lambda item: item.item_value + item.item_length, reverse=True)

time_end = time.time()  # 记录结束时间
time_sum = time_end - time_start  # 计算的时间差为程序的执行时间，单位为秒/s
print(time_sum)

best_solution
res_df = pd.DataFrame(columns=["原片材质", "原片序号", "产品 id", "产品 x 坐标", "产品 y 坐标", "产品 x 方向长度", "产品 y 方向长度"])
for res_item in res_items:
    res_df = res_df.append({"原片材质": res_item.item_material,
```

```
                                "原片序号": res_item.mate_id,
                                "产品 id": res_item.item_id,
                                "产品 x 坐标": res_item.x,
                                "产品 y 坐标": res_item.y,
                                "产品 x 方向长度": res_item.item_length,
                                "产品 y 方向长度": res_item.item_width },
                               ignore_index=True)

    res_df.to_csv(datapath+'_cut_program.csv', index=False, encoding='gbk')

    # fig, ax = plt.subplots(1, 1)
    #
    # rect = plt.Rectangle((0, 0), plate_width/plate_width, plate_length/plate_length, linewidth=2,
edgecolor='r', facecolor='none')
    # ax.add_patch(rect)
    # ax.set_xticks([])
    # ax.set_yticks([])
    # rect = plt.Rectangle((0, 0), 0.5, 0.5, linewidth=2, edgecolor='r')
    # ax.add_patch(rect)
    # plt.show()
    # currentAxis = fig.gca()

    # for bin_num in range(10):
    # fig, ax = plt.subplots(1, 1)
    # ax.set_aspect(1.0*plate_length/plate_width)
    # rect = plt.Rectangle((0, 0), plate_width / plate_width, plate_length / plate_length, linewidth=2,
edgecolor='r',
    #                      facecolor='none')
    # ax.add_patch(rect)
    # ax.set_xticks([])
    # ax.set_yticks([])
    # ax.set_title(bin_num)

    def drawout(best_solution, datapath):
        for index in range(len(best_solution)):
        # for index in range(85,88):

            fig, ax = plt.subplots(1, 1)

            ax.set_aspect(1.0 * plate_length / plate_width)
            rect = plt.Rectangle((0, 0), plate_width / plate_width, plate_length / plate_length,
                                 linewidth=2, edgecolor='silver',
                                 facecolor='none')
            ax.add_patch(rect)
```

```
            ax.axis('off')
            ax.set_xticks([])
            ax.set_yticks([])
            ax.set_title(index)
            for stripe in best_solution[index].strips:

                for stack in stripe.stack:
                    ax.plot(
                        [1.0 * (stack.width + stack.items[0].x) / plate_width,
                         1.0 * (stack.width + stack.items[0].x) / plate_width]
                        ,
                        [1.0 * (stack.items[0].y) / plate_length, 1.0 * (stack.items[0].y + stripe.height) / plate_length],
                        color='b'
                    )
                    for item in stack.items:
                        rect = plt.Rectangle((1.0 * item.x / plate_width, 1.0 * item.y / plate_length),
                                             1.0 * item.item_width / plate_width,
                                             1.0 * item.item_length / plate_length,
                                             edgecolor='black', facecolor='none')
                        ax.add_patch(rect)

                        ax.plot(
                            [1.0 * (item.x) / plate_width,
                             1.0 * (item.item_width + item.x) / plate_width]
                            ,
                            [1.0 * (item.y + item.item_length) / plate_length,
                             1.0 * (item.y + item.item_length) / plate_length],
                            color='g'
                        )

                        xoffset = 1.0*item.item_width/plate_width/2
                        yoffset = 1.0*item.item_length/plate_length/2
                        ax.text(1.0 * item.x / plate_width + xoffset, 1.0 * item.y / plate_length + yoffset,
                                "%d" % item.item_id , size=5)

                ax.plot([0, 1.0],
                        [1.0 * (stripe.height + stripe.y) / plate_length, 1.0 * (stripe.height + stripe.y) / plate_length],
                        color='r'
                        )
            plt.savefig(f"outputimg/{datapath}/{datapath}_{index}.png", bbox_inches='tight', dpi=200)
```

```
            # plt.show()
            plt.clf()

    drawout(best_solution, datapath)
    print(all_area / (len(best_solution)*plate_length*plate_width)) #  求解板材利用率
```

问题 2 代码	代码功能： 求解问题二的组批排样

```
import json

import pandas as pd
from FFFA import first_fit
# from Math.taskB.FFFA import first_fit
data_info = 'dataB4'
df = pd.read_csv("子问题 2-数据集 B/"+data_info +'.csv')

max_item_num = 1000
# m"2
max_item_area = 250e6
print(max_item_area)
class Res_item:
    def __init__(self, item_id, item_material, item_num, item_length, item_width, item_order):
        self.item_id = item_id
        self.item_material = item_material
        self.item_num = item_num
        self.item_length = item_length
        self.item_width = item_width
        self.item_order = item_order
        self.item_area = item_length*item_width

    def set_item_pos(self, mate_id, x, y):
        self.mate_id = mate_id
        self.x = x
        self.y = y

class Order:
    # 订单号
    def __init__(self, item_order):
        self.item_order = item_order

    def __len__(self):
        return len(self.items)
    # 订单里的 item
```

```
    def set_items(self, items):
        self.items = items

    def set_area(self, area):
        self.area = area

class OrderGroup:
    def __init__(self,id):
        self.id = id
        self.orders = []
    def __add__(self, other):
        self.orders.append(other)

class Cluster_Feature:
    def __init__(self, order_id):
        self.order_id = order_id
        self.orders_num = []
        self.order_area = []
import matplotlib.pyplot as plt
import matplotlib.patches as patches
plate_length = 2440
plate_width = 1220

def drawout(best_solution,data_info, batchpath,    materialpath):
    for index in range(len(best_solution)):
    # for index in range(85,88):

        fig, ax = plt.subplots(1, 1)

        ax.set_aspect(1.0 * plate_length / plate_width)
        rect = plt.Rectangle((0, 0), plate_width / plate_width, plate_length / plate_length,
                             linewidth=2, edgecolor='silver',
                             facecolor='none')
        ax.add_patch(rect)
        ax.axis('off')
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(index)
        for stripe in best_solution[index].strips:

            for stack in stripe.stack:
                ax.plot(
                    [1.0 * (stack.width + stack.items[0].x) / plate_width,
                     1.0 * (stack.width + stack.items[0].x) / plate_width]
```

```
                    ,
                    [1.0 * (stack.items[0].y) / plate_length, 1.0 * (stack.items[0].y + stripe.height)
/ plate_length],
                    color='b'
                )
                for item in stack.items:
                    rect = plt.Rectangle((1.0 * item.x / plate_width, 1.0 * item.y / plate_length),
                                         1.0 * item.item_width / plate_width,
                                         1.0 * item.item_length / plate_length,
                                         edgecolor='black', facecolor='none')
                    ax.add_patch(rect)

                    ax.plot(
                        [1.0 * (item.x) / plate_width,
                         1.0 * (item.item_width + item.x) / plate_width]
                        ,
                        [1.0 * (item.y + item.item_length) / plate_length,
                         1.0 * (item.y + item.item_length) / plate_length],
                        color='g'
                    )

                    xoffset = 1.0*item.item_width/plate_width/2
                    yoffset = 1.0*item.item_length/plate_length/2
                    ax.text(1.0 * item.x / plate_width + xoffset, 1.0 * item.y / plate_length +
yoffset,
                            "%d" % item.item_id , size=5)


            ax.plot([0, 1.0],
                    [1.0 * (stripe.height + stripe.y) / plate_length, 1.0 * (stripe.height + stripe.y) /
plate_length],
                    color='r'
                    )
        plt.savefig(f"outputimg/{data_info}/{batchpath}__{materialpath}_{index}.png",
bbox_inches='tight', dpi=200)
        # plt.show()
        plt.clf()
    # for res_item in res_items:
    #     if(res_item.mate_id==bin_num):
    #         rect = plt.Rectangle((1.0*res_item.x/plate_width,1.0*res_item.y/plate_length),
1.0*res_item.item_width/plate_width, 1.0*res_item.item_length/plate_length,
    #                              edgecolor='black', facecolor='none')
    #         ax.add_patch(rect)
    #         offset = 0.01
    #         ax.text(1.0*res_item.x/plate_width + offset,1.0*res_item.y/plate_length + offset,
```

```
"%d"%res_item.item_id)
        #                # plt.show()
        #
        #
        # plt.show()
# drawout(best_solution, datapath)

df_group = df.groupby('item_order')



all_orders = []
all_area = 0
for order in df_group:
        cur_order = Order(order[0])

        # cur_order.set_orders(order[1])
        res_items = []
        cur_area = 0
        for index, row in order[1].iterrows():
                # 同一宽要小，长要大
                if row.item_length > row.item_width:
                        res_item = Res_item(row.item_id, row.item_material, row.item_num, row.item_length,
row.item_width,
                                                row.item_order)
                else:
                        res_item = Res_item(row.item_id, row.item_material, row.item_num, row.item_width,
row.item_length,
                                                        row.item_order)
                res_items.append(res_item)
                cur_area += res_item.item_area
        all_area += cur_area
        cur_order.set_items(res_items)
        cur_order.set_area(cur_area)
        all_orders.append(cur_order)

labels = []
materials = set()
all_order_dict = []
for order in all_orders:
        order_dict = {"order_id": order.item_order,
                                "order_num": len(order),
                                "order_area": order.area}
        cur_area = 0
```

```
        for item in order.items:
            if item.item_material not in order_dict.keys():
                order_dict[item.item_material] = 1
            else:
                order_dict[item.item_material] += 1
            # if item.item_material not in order_dict.keys():
            #     order_dict[item.item_material] = 1
            materials.add(item.item_material)
        all_order_dict.append(order_dict)

labels.extend( materials )
labels.extend( ['order_id','order_num', 'order_area'] )
data = []
for order_dict in all_order_dict:
        order_data = []
        for label in labels:
            if label in order_dict.keys():
                order_data.append(order_dict[label])
            else:
                order_data.append(0)
        data.append(order_data)
import math
import numpy as np


def euler_distance(point1, point2) -> float:
        distance = np.sum(np.logical_xor(point1, point2))
        return distance



class ClusterNode(object):
        def __init__(self, vec, left=None, right=None, distance=-1, id=None, count=1, area=0,
item_num=0):
            self.vec = vec
            self.left = left
            self.right = right
            self.distance = distance
            self.id = id
            self.count = count
            self.area = area
            self.item_num = item_num
            self.min_val = np.min(vec[np.nonzero(vec)])
```

```
def new_distance(node1:ClusterNode, node2:ClusterNode):
    distance = 1e6
    for a, b in zip(node1.vec, node2.vec):
        if a == node1.min_val and b!=0:
            distance = -b

    return distance

class Hierarchical(object):
    def __init__(self, ):
        self.labels = None
    def fit(self, x):
        # orderlist = [int(item[130][5:]) for item in x]
        # numlist = [int(item[131]) for item in x]
        # arealist = [float(item[132]) for item in x]
        x_size = len(x[0])
        orderlist = [int(item[x_size - 3][5:]) for item in x]
        numlist = [int(item[x_size - 2]) for item in x]
        arealist = [float(item[x_size - 1]) for item in x]

        data = [item[:-3] for item in x]
        data = np.stack(data)
        nodes = [ClusterNode(vec=v, id=orderlist[i], item_num=numlist[i], area=arealist[i]) for i,v in
enumerate(data)]

        distances = {}
        point_num, future_num = np.shape(x)
        self.labels = [ -1 ] * point_num
        currentclustid = -1
        while True:
            min_dist = math.inf

            nodes_len = len(nodes)
            closest_part = None
            mval_id = sorted(range(len(nodes)), key=lambda k: nodes[k].min_val, reverse=False)



            if min_dist==math.inf:
                for i in range(nodes_len - 1):
                    for j in range(i + 1, nodes_len):

                        if(nodes[i].item_num+nodes[j].item_num>max_item_num or
```

```
                                nodes[i].area+nodes[j].area>max_item_area):
                            continue

                        d_key = (nodes[i].id, nodes[j].id)
                        if d_key not in distances:
                            distances[d_key] = euler_distance(nodes[i].vec, nodes[j].vec)
                        d = distances[d_key]
                        if d < min_dist:
                            min_dist = d
                            closest_part = (i, j)


        # min_node = None
        # min_val = -1
        # for i in range(nodes_len):
        #         tmp_vec = nodes[i].vec
        #         tmp_min = np.min(tmp_vec[np.nonzero(tmp_vec)])
        #         if min_val==-1 or tmp_min<min_val:
        #             min_val = tmp_min
        #             min_node = nodes[i]
        # for i in range(nodes_len):
        #         nodes[i]

        if min_dist == math.inf:
            break
        part1, part2 = closest_part
        node1, node2 = nodes[part1], nodes[part2]

        new_vec = node1.vec + node2.vec
        new_node = ClusterNode(vec=new_vec,
                               id=currentclustid,
                               count=node1.count + node2.count,
                               item_num=node1.item_num+node2.item_num,
                               area=node1.area + node2.area,
                               left=node1,
                               right=node2,
                               distance=min_dist,
                               )

        currentclustid -= 1
        del nodes[part2], nodes[part1]
        nodes.append(new_node)
    self.nodes = nodes
    self.calc_label()
```

```
    def calc_label(self):
        for i, node in enumerate(self.nodes):
            self.leafTraversal(node, i)

    def leafTraversal(self, node: ClusterNode, label):
        if node.left == None and node.right == None:
            self.labels[node.id-1] = label
        if node.left:
            self.leaf_traversal(node.left, label)
        if node.right:
            self.leaf_traversal(node.right, label)


# iris = datasets.load_iris()
#  计算程序运行时间
import time
time_start = time.time()  #  记录开始时间

my = Hierarchical()
my.fit(data)
res_orders = []
print(np.array(my.labels))
for og_id in range(max(my.labels)+1):
    cur_og = OrderGroup(og_id)
    res_orders.append(cur_og)

for order_id, label in enumerate(my.labels):
    str_order_id = 'order'+str(order_id+1)
    for order in all_orders:
        if order.item_order == str_order_id:
            res_orders[label].__add__(order)

# all_orders = sorted(all_orders, key = lambda order: len(order), reverse=True)
#
# #  按照订单种类计算距离
# cur_item_num = 0
# # m"2
# cur_item_area = 0
# cur_id = 0
# #  先从前往后分配
# is_vis = [False for _ in all_orders]
# res_orders = []
# while False in is_vis:
```

```
#         order_list = OrderGroup(cur_id)
#         cur_item_num = 0
#         # m"2
#         cur_item_area = 0
#         for index, order in enumerate(all_orders):
#                 if len(order)+cur_item_num <= max_item_num and order.area + cur_item_area < max_item_area and not is_vis[index] :
#                     is_vis[index] = True
#                     order_list.__add__(order)
#                     cur_item_num = len(order) + cur_item_num
#                     cur_item_area = order.area + cur_item_area
#         res_orders.append(order_list)
#         cur_id += 1


res_df = pd.DataFrame(columns=["批次序号", "原片材质", "原片序号", "产品 id", "产品 x 坐标", "产品 y 坐标", "产品 x 方向长度", "产品 y 方向长度"])
cur_len = 0
batch_order = {}
for order_group in res_orders:
    batch_id = order_group.id
    batch_order[batch_id] = []
    res_items_group = []
    for order_item in order_group.orders:
        batch_order[batch_id].append(order_item.item_order)
        res_items_group.extend(order_item.items)
    res_items_dict = {}
    # 对 group 中的 订单 item 根据材料分组
    for res_item in res_items_group:
        res_items_dict.setdefault(res_item.item_material,[]).append(res_item)
    # 对每个材料进行切割
    solution = []
    for material in res_items_dict.keys():
        cur_items = res_items_dict[material]
        cur_items = sorted(cur_items, key=lambda item: [item.item_length, item.item_width], reverse=True)

        cur_solution = first_fit(cur_items)
        for cur_item in cur_items:
            solution = {"批次序号": batch_id, "原片材质": material,"原片序号": cur_item.mate_id,"产品 id": cur_item.item_id,"产品 x 坐标":cur_item.x,
                        "产品 y 坐标":cur_item.y, "产品 x 方向长度": cur_item.item_width, "产品 y 方向长度":cur_item.item_length}
            res_df = res_df.append(solution,
```

```
                                        ignore_index=True)
            cur_len += len(cur_solution)
            drawout(cur_solution,data_info, batch_id, material)
time_end = time.time()  # 记录结束时间
time_sum = time_end - time_start  # 计算的时间差为程序的执行时间，单位为秒/s

# import csv
# with open('test.csv', 'w') as f:
#         for key in batch_order.keys():
#             f.write("%s,"%(key))
#             for order in batch_order[key]:
#                 f.write("%s" % order.replace("\"",""))
#                 f.write("、")
#             f.write("\n")

print(time_sum)
print(cur_len)
print(all_area / (cur_len*2440*1220))
res_df.to_csv(data_info+"_sum_order.csv", encoding='gbk',index=False)
```