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	湘潭大学
参赛队号	22105300030
队员姓名	1.黄铭杰
	2.张慧芳
	3.稂昱星

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　　方形件组批优化问题研究

摘　　要：

本文通过采用迭代法、K-Means 聚类分析法求解混合整数规划模型，对方形件的排布优化和组批问题进行研究。

在问题一中，将板材的切割问题视为一个项目件的堆叠问题。首先以最小原片耗材量为目标函数，建立给定约束条件的混合整数规划模型Ⅰ。规定每个项目件必须完成一次堆叠，堆叠的高度必须合理，因此设置虚拟变量α、β、γ来判断 item、stack、stripe、bin 是否依次存在被包含关系，限定项目件的堆叠高度不应超过栈的高度，堆栈的宽度不得超过原片的宽度，条带的高度不能超过原片的高度。在在实际应用中，以 item 的高度 h 作为排样依据，放入模型Ⅰ中求得虚拟变量参数值α、β、γ。由于 stack 的数值指标未知，设置调整参数 H、均值、中位数对模型进行优化，最终，发现采用高度进行排序、均值、中位数作为调整参数的模型拟合效果最好，在数据集 A 中的平均板材利用率达到了 94.06%。

在问题二中，需要考虑在订单号不可分批与板材材质存在差异的基础上，解决数据量较大时订单件有约束条件的组批问题。首先将相同的订单号放在一起，设定与材质相关的指标构建矩阵作为 K-Means 聚类的依据，同时约束单个批次产品项总数 1000 件和单个批次产品面积总上限 250 平方米。通过欧氏距离和类平均距离来划分簇族，找到符合分类要求的最小类别数 K, 并不断调整超参数 max_order、min_order、K，直至找到最优可行解，完成产品项分批。将产品项分批完成后，对每一个批次中不同材质的产品项进行分组，对每一组单独采用模型Ⅰ中的排样优化方法，最终得到输出结果为 83.32%。

本文问题一中的迭代法求解代码未调用第三方库，全部自行写入封装，具有运算速度快、可修改性强的特点。本文中对方形件的订单组批以及排样优化建模求解的效果均较为稳健，对箱包问题、下料问题可提供一定指导意义，对于 PCB 电路板、板氏家具、3C 家电等领域的实际应用具有一定参考价值。

关键词：迭代法；K-Means 聚类；超参数；堆叠.

一、 问题重述

在智能制造的时代背景下，机器制造业、航空航天业、建筑业等各个领域的设施更新迭代迅速、产品变化灵活多样，个性化定制需求量的上升成为必然。方形件作为制造业中必不可少的一类产品，其个性化与高效率生产之间的矛盾若能被解决，对推动整个订单批次的生产将会有极大意义。将方形件产品的排布优化视为一个下料问题，在只考虑齐头切的切割工艺以及精确排样的基础上，合理地利用耗材，降低生产成本，寻找最优的切割方法。

就给定的产品相关信息，在尽量减少板材用量的基础上，建立混合整数规划模型，求解以下问题：

问题 1：就给定的原片 1220*2440 进行三阶段切割，将板材分割成 stripe（条带），条带分割成 stack（栈），最后再将栈分割成 item（产品项），要求在同一栈中，最后切割的产品项与相邻的产品项长度或宽度至少一个相同，且产品项必须结构完整，不可由拼接构成。

根据数据集中给定的订单要求进行切割，在板材用量尽可能少的情况下，找到原片的最优切割方法。

问题 2：在问题 1 的基础上，建立混合整数模型，将每个 dataBi.csv 中的订单构建成多种批次，要求相同的订单不能被拆分于不同批次中，每个批次产品项的总数不能超过 1000，每个批次的面积总和不能超过 250 平方米，且同一块原片只能匹配一种材质的产品项，在此基础上，对每个批次的产品项进行独立排样。

在板材用量尽可能少的情况下，根据订单的实际需求获得更优的订单组合，降低原材料的使用成本，提高生产效率，找到每个批次的最优切割方法。

二、 问题分析

2.1 问题一的分析

问题一要求建立混合整数规划模型，对板材的排布进行优化，求得尽可能小的板材原片消耗量。为达到这一目标，我们将最小的原片耗材量视为目标函数，在充分考虑约束条件的基础上，寻找 bin（原片）、stripe（条带）、stack（栈）、item（产品项）之间存在的关系，通过解读各个切割阶段之间的具体联系，设定与之匹配的参数，从而构建与本题目标相关的方程组。

采用逆推的思维方法，把原片的切割问题看成是订单件的组合问题，将需要的订单件进行堆叠，item 组合成 stack、stack 组合成 stripe、再将 stripe 进行堆叠组合，补足部分余料后，最终构成题目所设定的原片。

在确定模型以后，首先考虑使用迭代法，通过不断的输入订单需求，寻求局部最优解。

2.2 问题二的分析

在问题一的基础上，问题二增加了新的约束条件，一是关于批次项的限定条件，要求每个批次产品项数量不得大于 1000 且单个批次产品项的总面积不得大于 250 平方米，二是给出订单号和材质作为批次的划分依据，要求同一份订单不能被拆分为不同批次，且每个订单的材质可能存在差异。这要求我们先将订单按照约束条件进行分类，再对分类后的各个批次单独进行排样优化。

若将排样优化视为一个主问题，则组批问题可视为一个三阶段排样的子问题。面

对订单的组批问题[1]，我们考虑先将产品件按订单号归为一类，再进行有约束条件的K-Means 聚类，在 K-Means 聚类的过程中，为了使板材原片的利用率较高，要尽可能使相同材质的在同一组批中。完成聚类后，将每一类归为一个批次，对每个批次中不同材质的产品项采用问题一中的混合整数规划模型进行迭代，最终求出每个批次的近似解[2]。

三、 模型假设

1. 原片规格统一且材质均匀分布，没有杂质区域；
2. 切割工艺采用齐头切方式；
3. 成品件互相独立，各产品项没有交叠[3]；
4. 切割得到的产品完整，不可由拼接而成；
5. 第一刀与原片短边的水平方向平行；
6. 每一刀总与上一刀的切割方向垂直[4]；
7. 最多只考虑三阶段切割；
8. 排样方式只考虑精确排样；

四、 符号说明

	符号	说明
下标	l	原片编号
	k	Stripe 编号
	j	Stack 编号
	i	Item 编号
变量	h	高度
	w	宽度
	d	欧式距离
	D	类平均距离
	$a_m^{(n)}$	订单 n 中有材质 m 的项目件个数
	b_n	订单 n 包含项目件的个数
	z	耗用原片数
	M_{item}^{p}	第 p 个批次中产品项的数量
	S_{item}	第 i 块板材中产品项的面积
虚拟变量	$\alpha_{j,i}$	item 是否在 stack 中
	$\beta_{k,j}$	Stack 是否在 stripe 中
	$\gamma_{l,k}$	Stripe 是否在 bin 中
常量	H	板材原片高
	W	板材原片宽

五、 模型的建立与求解

5.1 问题一：混合整数规划模型的排样优化

5.1.1 模型建立

在本题中，将板材原片的分割视为一个三阶段排样[5]，第一阶段为平行于原片短边的水平切割，剪切出一组 stripe，第二阶段为垂直于短边的切割，将每个 stripe 切割成不同的 stack，第三阶段仍为水平切割，将 stack 切割为所需要的精确 item，给出切割模式的相关示例如图 1 所示：

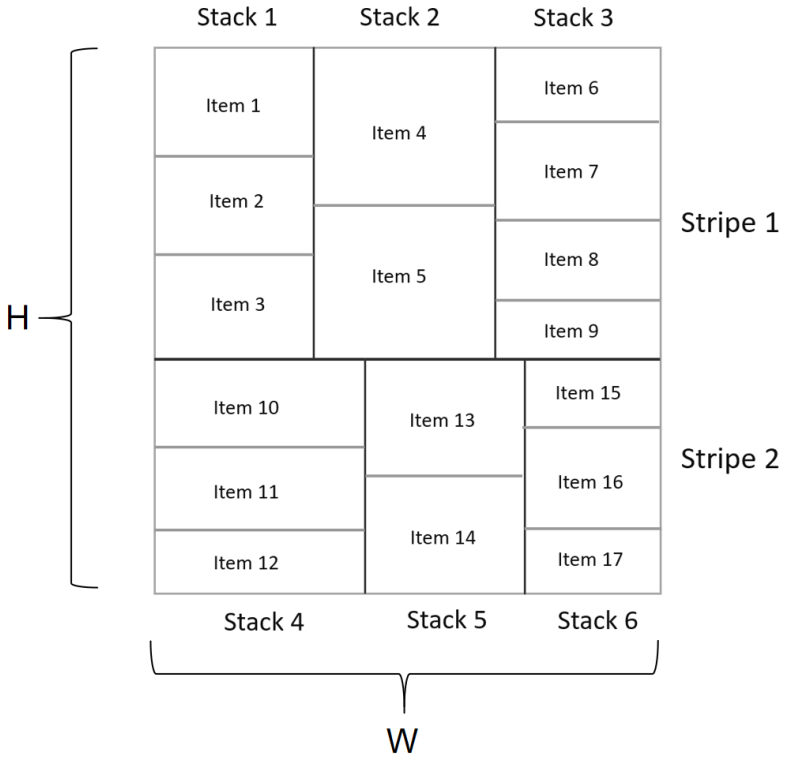


图 1 二维三阶段切割图

针对长宽为 $H \times W$ 的板材原片确定最优排样方案[6]，引入 i、j、k 分别表示 item、stack、stripe 的编号,假设在每种排样方式中引入 n 个 item，可视第 i 个 item 的宽度和高度为 (w_i, h_i),第 j 个 stack 的宽高为 (w_j, h_j)，以此类推，则第 k 个 stripe 的宽高为 (W, h_j)，对于合理的板材分割模型[7]，$0 < w_i \le W, 0 < h_i \le h_j \le h_k \le H$ 显然成立。

假设虚拟变量 $\gamma_{l,l}$ 表示原片的切割情况，若某块原片被切割，则其数值为 1，若原片未被切割，则数值为 0，要使耗用原片数量最小，即令 $\gamma_{l,l}$ 求和最小，将目标函数表示为公式 5-1，构建模型 I [8]：

$$\min \quad z = \sum_{l=1}^{n} \gamma_{l,l}. \tag{5.1.1}$$

为更好的实现目标函数，我们从两个角度对函数的约束条件进行考虑，一是判断项目堆叠是否成功放置，保证每个项目件必须完成一次堆叠，二是判断项目高度是否合理，若模型成立，项目件的堆叠高度不应超过栈的高度，堆栈的宽度不得超过原片的宽度，条带的高度不能超过原片的高度。

根据产品件的高度 h 将 item 进行排序，选择 item 从大到小进行堆叠，将不符合该原片组合的 item 暂时放置在下一次后，最终确定 n 个 item 近似地组合成某板材原片。在每一块原片中，以 item 的最小索引作为 stack 的编号，同理，以一个条带中 stack 最高的高度，也即最大的 h_j 对应的索引作为 stripe 的编号 k。

为更好的确定产品件的堆叠成立，设定对应的虚拟变量 α、β、γ，当 item i 落在在 stack j 中，可令虚拟变量 $\alpha_{j,i}$ 的值为 1，当 stack j 落在 stripe k 中，令 $\beta_{k,j}$ 为 1，同样的，若 stripe k 被包含在 bin l 中，则令虚拟变量 $\gamma_{l,k}$ 值为 1。在实际实现中，还要考虑到产品件的堆叠是否合理，也即一个栈中 item 的堆叠高度是否小于 stack 的高度，一个条带中 stack 的堆叠宽度是否小于 stripe 的宽度，若答案为否，则认为该堆叠不成立。

基于此，给定约束条件：

$$\sum_{j=1}^{i}\alpha_{j,i}=1, i=1,\cdots,n, \tag{5.1.2}$$

$$\alpha_{j,i}=0, j=1,2,\cdots,n\text{且}\forall \mathrm{i}<\mathrm{j}\left|w_i\neq w_j\cup\sum_{i=j}^{n}h_i\alpha_{j,i}>\sum_{k=1}^{j}h_j^{'}\beta_{k,j}\right., \tag{5.1.3}$$

$$\sum_{k=1}^{j}\beta_{k,j}=1, k=1,\cdots,n, \tag{5.1.4}$$

$$\sum_{l=1}^{k}\gamma_{l,k}=1, l=1,\cdots,n, \tag{5.1.5}$$

$$\sum_{i=j}^{n}h_i\alpha_{j,i}\le\sum_{k=1}^{j}h_j^{'}\beta_{k,j}, j=1,\cdots,n, \tag{5.1.6}$$

$$\sum_{j=k}^{n}w_j\beta_{k,j}\le W, k=1,\cdots,n, \tag{5.1.7}$$

$$\sum_{k=l}^{n}h_k^{''}\gamma_{l,k}\le H, l=1,\cdots,n. \tag{5.1.8}$$

其中：

$$\begin{cases}\alpha_{j,i}\in\{0,1\}, j=1,2,\cdots,n, i=j,\cdots,n;\\ \beta_{k,j}\in\{0,1\}, k=1,2,\cdots,n, j=k,\cdots,n;\\ \gamma_{l,k}\in\{0,1\}, l=1,2,\cdots,n, k=l,\cdots,n.\end{cases} \tag{5.1.9}$$

5.1.2 模型求解

迭代法[9]主要应用于较难直接确定具体解的情况，从一个初始值出发，通过递推公式和循环算法，来不断逼近于近似解。对于混合整数规划模型，往往难以直接定解，因而本文采用基于约束条件的迭代搜索算法来求解。

算法步骤：

Step1：将各项目件根据高度 h 进行排序；

Step2：依次放入各 item，每确定一个项目件的放置情况，求得对应参数

$\alpha_{j,i}$、$\beta_{k,j}$、$\gamma_{l,k}$的数值；

Step3：对模型进行调整优化，确定调整后的 stack 数值指标，判断约束条件是否成立；

Step4：当约束条件成立，则输出近似解，约束条件不成立，则继续迭代，直至所有订单完成排样。

根据算法迭代绘制排样算法流程图如图 2 所示：

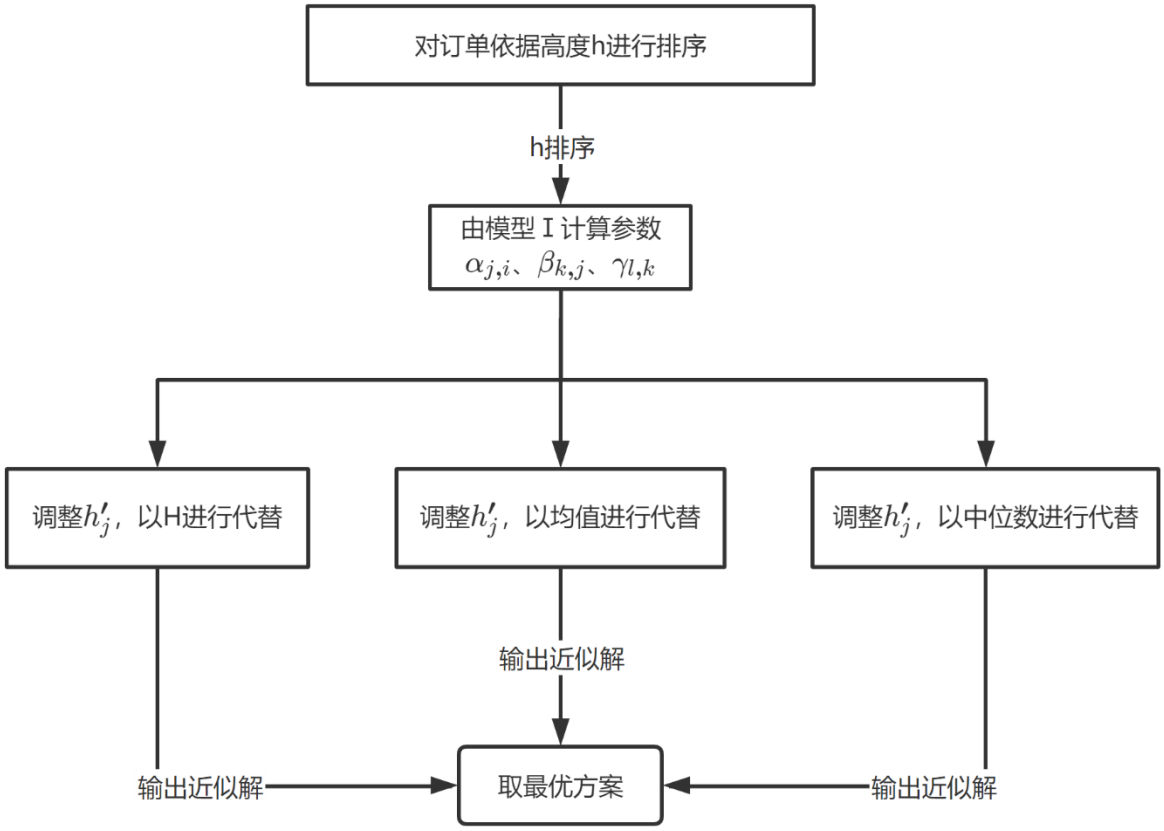


图 2 排样优化算法流程图[10]

5.1.3 模型求解结果及分析（算法的输出效果图见附录 5）

本文采用的设备为 cpu：intel i5-7300HQ，内存为 8g，操作系统为 windows10，未用 GPU 加速计算，用 conda 4.8.3 版本创建解释器为 python 3.8 的虚拟环境求解。我们发现无论是其他算法亦或是迭代法均难以求解h'_j，因此我们采用超参数来代替h'_j，为更好的比较不同超参数对模型优化效果的影响，我们根据具体情况，将 H、均值以及中位数作为调整后的超参数分别建立模型，得到输出结果如表 1：

表 1 超参数输出效果对比表

	板材利用率（%）	运行时间（s）	耗用原片数（块）
H	91.22	190.92	364
平均数	93.53	188.73	355
中位数	94.06	179.19	353

在对多种排样方案的输出效果进行对比后，我们发现相较于其他两种调整替代，中位数作为调整后的h'_j模型拟合效果最好，板材利用率显著提高，运行时间缩短，耗

用原片数最少，选取部分排样方案展示如下：

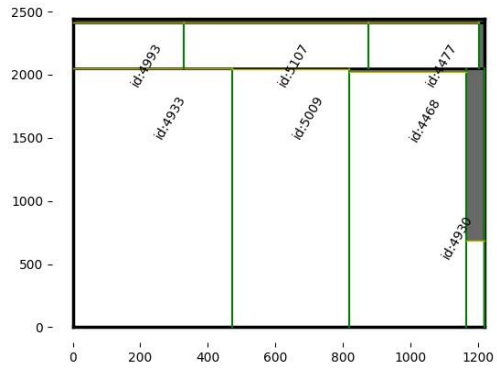


图 3 A1 bin5

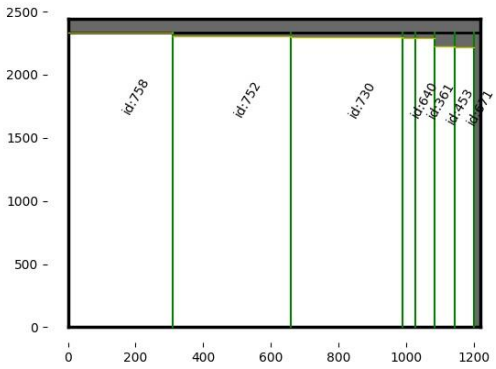


图 4 A2 bin30

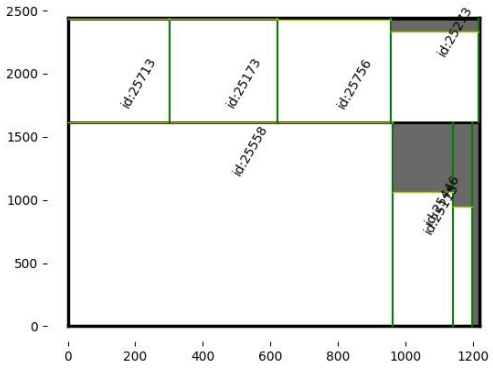


图 5 A3 bin55

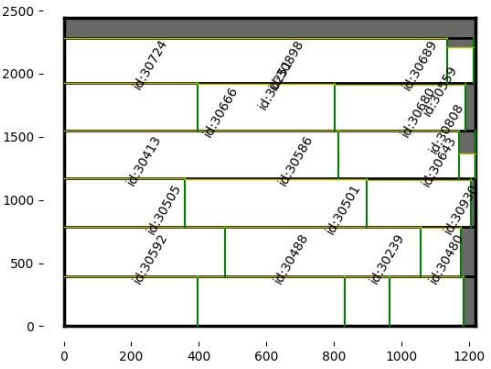


图 6 A4 bin80

表 2 dataA 结果分析表

结果指标	data A1	data A2	data A3	data A4	data A
板材利用率（%）	93.87	93.12	94.08	95.18	94.06
板材原片数（块）	89	89	89	86	353
运行时间（s）	46.26	43.22	45.53	44.18	179.19

不难发现，根据上文的算法，选用中位数作为超参数，数据集 A 的板材利用率达到了 93%以上，每个 data 板材原片数均低于 90 块且在 50 秒内完成求解，模型表现效果稳健。

5.2 问题二：混合整数规划模型的组批与排样优化

5.2.1 模型建立

针对本问题中项目件的分批问题，先将相同订单的产品项放在一起，再根据方形件材质定义一个矩阵作为 K-Means 聚类的依据。假设一个 data 中有 n 个订单，m 种方形件材质，由于产品项的材质种类均为无序多分类变量，为使数据量化，在引入模型前对数据进行哑变量处理，将其赋值为 0、1、2…m-1。

定义矩阵 I 表示在不同的订单中，以某种材质的项目件需求个数 $a_i^{(j)}$ 占该订单所

有产品项个数b_j为指标距离用于衡量两个（订单材质）集合的区分度；定义矩阵Ⅱ表示在不同的订单中，以某种材质的项目件需求个数$a_i^{(j)}$作为聚类的分类依据。

基于此，建立聚类矩阵如表 3 和表 4：

表 3 矩阵Ⅰ

	order1	order2	order3	…	order n
材质 1	$\frac{a_1^{(1)}}{b_1}$	$\frac{a_1^{(2)}}{b_2}$	$\frac{a_1^{(3)}}{b_3}$	…	$\frac{a_1^{(n)}}{b_n}$
材质 2	$\frac{a_2^{(1)}}{b_1}$	$\frac{a_2^{(2)}}{b_2}$	$\frac{a_2^{(3)}}{b_3}$	…	$\frac{a_2^{(n)}}{b_n}$
材质 3	$\frac{a_3^{(1)}}{b_1}$	$\frac{a_3^{(2)}}{b_2}$	$\frac{a_3^{(3)}}{b_3}$	…	$\frac{a_3^{(n)}}{b_n}$
…	…	…	…	…	…
材质 m	$\frac{a_m^{(1)}}{b_1}$	$\frac{a_m^{(2)}}{b_2}$	$\frac{a_m^{(3)}}{b_3}$	…	$\frac{a_m^{(n)}}{b_n}$

表 4 矩阵Ⅱ

	order1	order2	order3	…	order n
材质 1	$a_1^{(1)}$	$a_1^{(2)}$	$a_1^{(3)}$	…	$a_1^{(n)}$
材质 2	$a_2^{(1)}$	$a_2^{(2)}$	$a_2^{(3)}$	…	$a_2^{(n)}$
材质 3	$a_3^{(1)}$	$a_3^{(2)}$	$a_3^{(3)}$	…	$a_3^{(n)}$
…	…	…	…	…	…
材质 m	$a_m^{(1)}$	$a_m^{(2)}$	$a_m^{(3)}$	…	$a_m^{(n)}$

欧式距离作为常用的距离测度单位，可来评估两类之间的相似程度，欧式距离越小表明其两类之间的距离越接近，欧氏距离大则说明两类之间距离较远，基于上文矩阵中所采用的指标，可选取欧式距离较小的类别将其归为一个簇类。

$$d(x,y)=\sqrt{(x-y)^T(x-y)}=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}. \tag{5.2.1}$$

采用类平均距离来计算订单簇之间的距离：

$$D_{HK}=\sqrt{\frac{1}{n_H n_K}\sum_{\substack{i\in H\\ j\in K}} d_{ij}^2}=\sqrt{\frac{n_I}{n_I+n_J}D_{HI}^2+\frac{n_J}{n_I+n_J}D_{HJ}^2}. \tag{5.2.2}$$

在进行 K-Means 聚类寻找最优 K 值时，采用轮廓系数法，该方法综合考虑了簇之间的密集性与分散性。当数据集被分割为理想的 K 个簇，则簇内样本会很密集，而簇间样本会很分散。轮廓系数的计算公式可以表示为：

$$s(i)=\frac{b(i)-a(i)}{\max(a(i),b(i))} \tag{5.2.3}$$

式中,， $a(i)$ 体现了簇内的密集性，代表样本 i 与同簇内其他样本点距离的平均值；反映了簇间的分散性，它的计算过程是，样本 i 与其他非同簇样本点距离的平均值，然后从平均值中挑选出最小值。

轮廓系数法的计算公式可以进一步写为：

$$s(i)=\begin{cases}1-\dfrac{a(i)}{b(i)}, a(i)<b(i);\\ 0, \quad a(i)=b(i);\\ \dfrac{b(i)}{a(i)}-1, a(i)>b(i).\end{cases} \tag{5.2.4}$$

$S(i)$ 接近于-1 时，说明样本 i 分配的不合理，需将其分配到其他簇中；当 $S(i)$ 近似为 0 时，说明样本 i 落在了簇的边界处；当 $S(i)$ 近似为 1 时，说明样本 i 的分配是合理的。

根据题目给定的约束条件确定目标函数：

$$\min \quad \sum_{p=1}^{k} z_p. \tag{5.2.5}$$

在模型Ⅰ的基础上补充约束条件:

$$M_{item}^{p}<1000 \tag{5.2.6}$$

$$\sum_{i=1}^{M_{item}} S_{item}^{i}<2.5\times10^{8} \tag{5.2.7}$$

$$S_{item}\le H\times W \tag{5.2.8}$$

最终建立混合整数规划模型Ⅱ，其中 z_p 表示第 p 个批次订单中耗用的板材数量， M_{item}^{p} 表示第 p 个批次中产品项的数量， S_{item}^{i} 表示第 i 块板材中产品项占用的面积，该值小于原片的最大面积 $H\times W$ 。

基于以上信息，可以进行带约束的 K-means 聚类，利用距离的远近将目标数据聚为指定的 K 个簇，进而使样本呈现簇内差异小，簇间差异大的特征。

具体步骤如下：

（1）从数据中挑选 K 个样本点作为原始的簇中心。

（2）基于约束条件将对象分配到最相似的簇中，计算剩余样本与簇中心的距离，并把各样本标记为离 K 个簇中心最近的类别。

（3）重新计算各簇中样本点的均值，并以均值作为新的 K 个簇中心。

（4）不断重复第二步和第三步，直到簇中心的变化趋于稳定（新的簇中心与原簇中心相等或小于指定阈值），形成最终的 K 个簇。

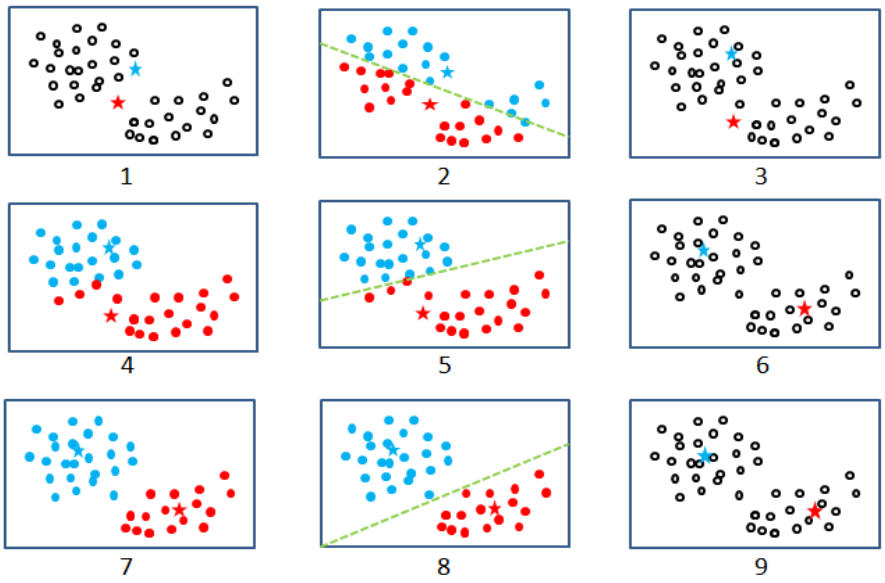


图 7 二维 K-Means 聚类图

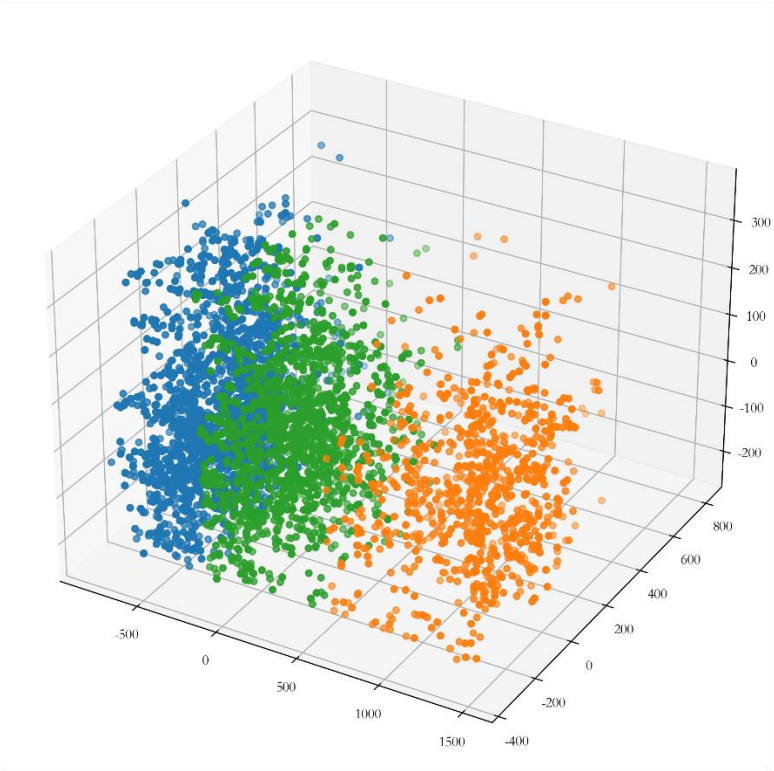


图 8 三维 K-Means 聚类图

5. 2. 2 模型求解

基于以上的混合整数规划模型建模结果，采用 conda 4.8.3 版本创建解释器为 python 3.8 的虚拟环境求解，对数据 dataB 进行组批以及排样优化：

步骤：

Step1：对订单与项目件的材质关系矩阵进行求解；

Step2：通过欧氏距离和类平均距离来划分簇族，从最小值 K 开始遍历，基于约束

条件下将对象分配到最相似的簇中，从而找到符合分类要求的最小类别数 K；

Step3：调整超参数 max_order、min_order、K，循环 Step2；

Step4：选择最优分类组合，确定具体批次划分；

Step5：对不同材质的产品项采用模型 I 中的排样优化方法，完成排样。

注：max_order 和 min_order 分别为每个批次中 order 的最大数量以及最小数量，由于题目对每个批次的订单数与订单面积有所约束，而我们以 order 作为初始类别，因而需要引入这两个指标作为间接约束。

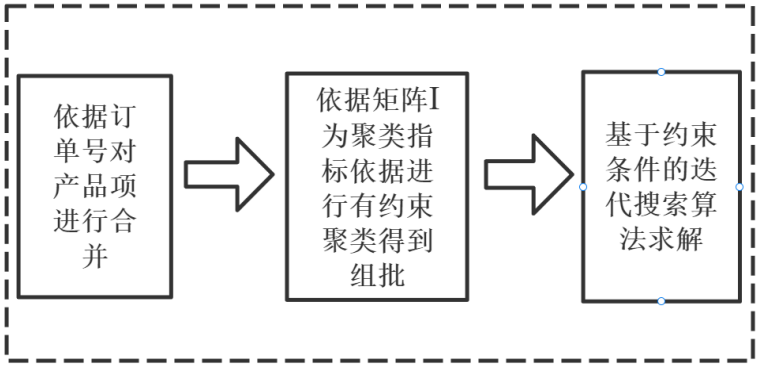


图 9 模型 II 算法流程图

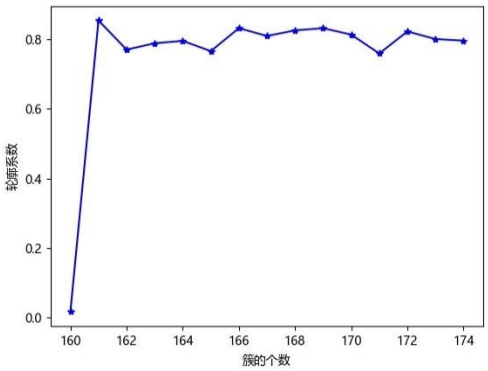


图 10 B1 与轮廓系数对应的折线图

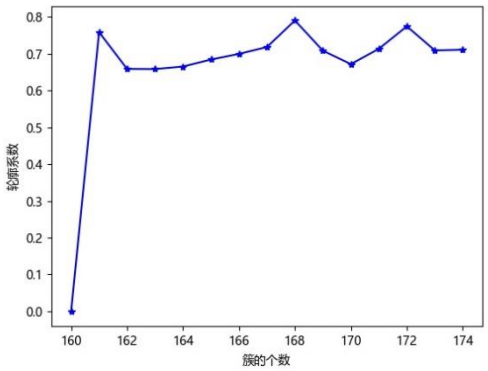


图 11 B2 与轮廓系数对应的折线图

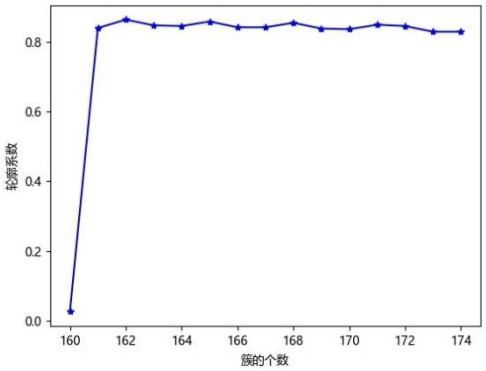


图 12 B3 与轮廓系数对应的折线图

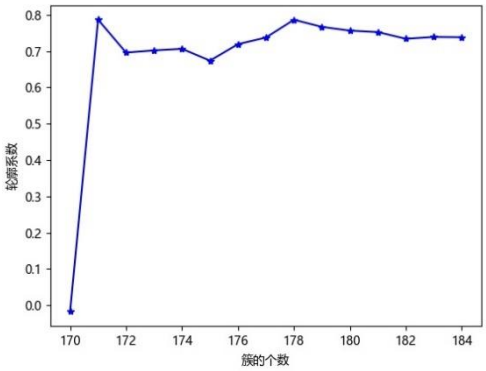


图 13 B4 与轮廓系数对应的折线图

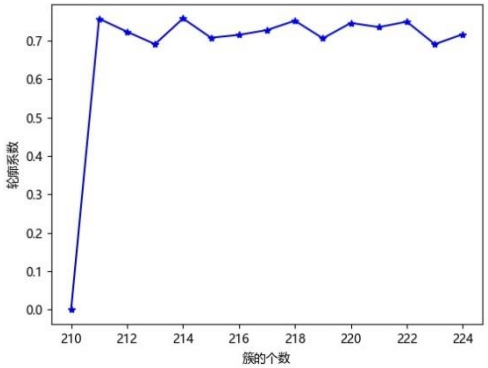


图 14 B5 与轮廓系数对应的折线图

通常来说，与轮廓系数相对应的折线图，轮廓系数越大，且越接近于 1，说明聚类效果越好。通过进行有约束的 K-means 聚类，得到与轮廓系数对应的折线图，发现每个 data B 的轮廓系数均逼近 0.8，说明项目件的分配较为合理，符合分类要求的最小类别数 K 分别为 161、161、161、171、211。

在此基础上，对超参数进行调整，找到满足约束条件的可行解并进行对比，最终确定局部最优解，并输出效果图如下。

5. 2. 3 模型结果分析

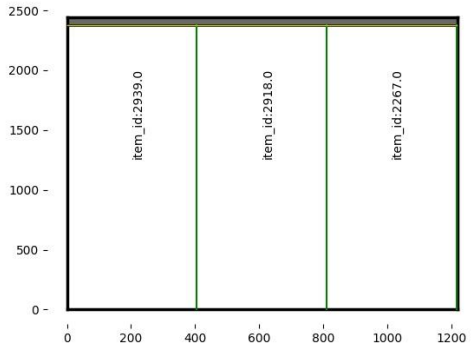


图 15 B1_1.0_119.0

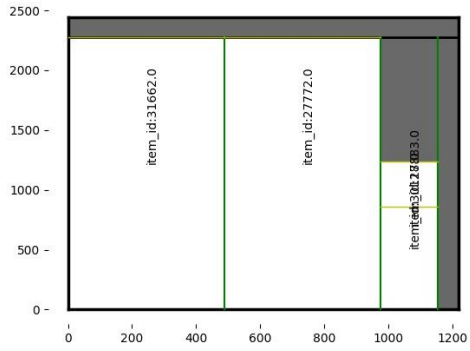


图 16 B1_2.0_114.0

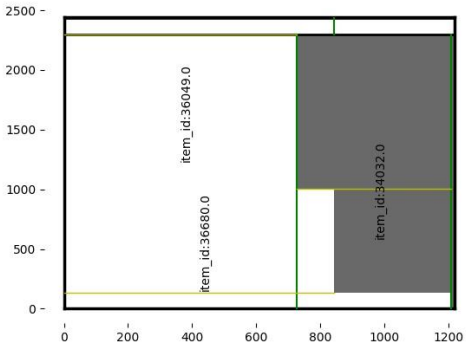


图 17 B1_3.0_86.0

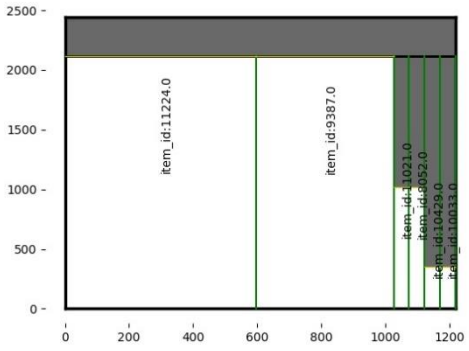


图 18 B1_4.0_119.0

表 5 data B 结果分析表

结果指标	data B1	data B2	data B3	data B4	data B5	data A
板材利用率（%）	84.35	82.67	82.36	83.68	83.53	83.32

根据数据输出结果分析表可以看出，在数据集 B 的多个 data 中，板材利用率始终维持在 80%以上，总体拟合效果较好，模型表现稳健。

六、 模型的评价、改进与推广

6.1 模型的优点

1.问题一代码未调用第三方库，运算速度快，仅需 179 秒即可完成整个数据集 A 的运算、输出和保存；

2.问题一中代码由自己进行写入封装，可修改性强；

3.通过设定超参数建立多种模型进行对比，有利于选择更优的模型，最终选定的数学模型在问题一中的平均板材利用率达到了 94.06%；

4.模型的排样方式采用精确排样，可在三阶段内完成分割，步骤较为简洁且板材利用率高。

5.问题一中，多次调整超参数，模型效果均较为理想，可见模型的适应性和稳定性均较强。

6.2 模型的缺点

1.只对三阶段的精确排样进行建模，并直接采用迭代法进行求解，未采用其他算法，如启发式算法等，无法进行对比验证。

2.排样方案图的输出可以继续优化，部分标签定位不够清晰精准。

6.3 模型的改进

可以考虑采用不同算法进行求解，同时，还可以考虑多种排样方式，如非精确排样方式，对输出结果进行对比验证，选择最佳方法；重新优化图片输出算法，改进定位问题。

6.4 模型的推广

本文中对方形件的订单组批以及排样优化的建模求解方法，对箱包问题、下料问题同样具有一定指导意义，并且可以在 PCB 电路板、板氏家具、3C 家电等领域发挥实际应用价值。

七、 参考文献

[1] 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.

[2] 孔令熠. 基于普通条带的二维多阶段排样算法[D].广西大学,2014.

[3] 李立平. 二维三阶段排样算法研究[D].广西大学,2016.

[4] 黄丹妮. 启发式算法求解二维矩形切割优化问题研究[D].华中科技大学,2019..

[5] 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.

[6] 齐中娟.适合“一刀切”剪切方式的矩形件排样算法[J].科技资讯,2014,12(16):95-96.DOI:10.16661/j.cnki.1672-3791.2014.16.073.

[7] 蔡正军,龚坚,刘飞.板材优化下料的数学模型的研究[J].重庆大学学报(自然科学版),1996(02):82-88.

[8] Jakob Puchinger, Günther R. Raidl,Models and algorithms for three-stage two-dimensional bin packing,European Journal of Operational Research,Volume 183, Issue 3,2007,Pages 1304-1327.

[9] 迭代法， https://baike.baidu.com/item/%E8%BF%AD%E4%BB%A3%E6%B3%95/10913188，2022 年 10 月 8 日.

[10] 徐宗煌,徐剑莆,李世龙,林慧雅,许美燕.运用启发式算法的木板最优切割方案[J].宁德师范学院学报(自然科学版),2021,33(01):11-20.

附录

附录 1

介绍：问题一的主代码

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function


import os
import time
import pandas as pd

from demo import Cycle_solution_A

    # 衡量指标
def metrics(data, number, max_length, max_width):
    """
    :parameter
    data:[[序号：index，订单 id:item_id，订单材料:item_material，订单所需要数量:item_num，材料长度:item_length，材料宽度:item_width，订单号:item_order]*size]
    number:消耗的总板材
    max_length:板材长
    max_width:板材宽
    """
    total_item = 0.
    for vector in data:
        total_item += vector[4] * vector[5]
    run_rate = total_item / (number*max_length*max_width)

    return run_rate


# 问题一
# 加载数据
file_A = []
for i in range(1, 5):
    file_path = os.path.join(r"E:\python document\challenge\2022 年 B 题\子问题 1-数据集 A", "dataA"+str(i)+".csv")
    if os.path.exists(file_path):
        file_A.append(pd.read_csv(file_path))

data_A_lis = []
avg_length = []
for i in file_A:
    avg_length.append(i['item_length'].mean())
    j = i.sort_values(by='item_length', ascending=False)
    j = j.values.tolist()
    data_A_lis.append(j)
```

```
# [序号：index，订单 id:item_id，订单材料:item_material，订单所需要数量:item_num，材料长度:item_length，材料宽度:item_width，订单号:item_order]

print(data_A_lis)
# 参数
max_length = 2440
max_width = 1220

path = r"E:\python document\challenge\2022 年 B 题\solution_A"
f_txt = open(r"E:\python document\challenge\2022 年 B 题\solution_A\min_n.txt", "w")

# 求解
min_n_data_A = []
rate_A_lis = []
run_time_lis = []
_time = time.time()

for A_index, data_set in enumerate(data_A_lis):
    data_size = len(data_set)
    paths = os.path.join(path, "A{0}".format(A_index+1))
    solution = Cycle_solution_A(data_size, paths, avg_length[A_index], max_length, max_width)
    min_n, cut_program = solution.cycle_com(data_set)
    cut_program.to_csv(r"E:\python document\challenge\2022 年 B 题\solution_A\A{0}\data_A{0}_cut_program.csv".format(A_index+1), index=None, encoding='utf_8_sig')
    rate = metrics(data_set, min_n, max_length, max_width)
    run_time_lis.append(time.time() - _time)
    min_n_data_A.append(min_n)
    rate_A_lis.append(rate)
    print("data_A{0}:finish".format(A_index+1))

f_txt.write("min_n_data_A:" + "\n")
f_txt.write(str(min_n_data_A) + "\n" + "\n")
f_txt.write("rate_A_lis:" + "\n")
f_txt.write(str(rate_A_lis) + "\n" + "\n")
f_txt.write("run_time:" + "\n")
f_txt.write(str(run_time_lis) + "\n" + "\n")

print(min_n_data_A)
print(rate_A_lis)
```

附录 2

介绍：问题一的迭代法实现

```
class Cycle_solution_A():
```

```
    def __init__(self, data_size, path, average_length, max_length=2440, max_width=1220):
        self.max_length = max_length
        self.path = path
        self.average_length = average_length
        self.max_width = max_width
        self.data_size = data_size

        # alpha、beta、gamma 的定义
        self.alpha = []
        self.beta = []
        self.gamma = []

        for j in range(data_size):
            self.alpha.append([0 if j > i else 1 for i in range(data_size)])
        for j in range(data_size):
            self.beta.append([0 if j > i else 1 for i in range(data_size)])
        for j in range(data_size):
            self.gamma.append([0 if j > i else 1 for i in range(data_size)])


    def cycle_com(self, data):
        """
        :parameter
        data:[[序号：index, 订单 id:item_id, 订单材料:item_material, 订单所需要数 量 :item_num, 材 料 长 度 :item_length, 材 料 宽 度 :item_width, 订 单 号:item_order]*size]
        """
        stack_lis, info_stack_lis, alpha = self.item_into_stack(data, self.alpha)
        stripe_lis, stripe_index_lis, info_stripe_lis, beta = self.stack_into_stripe(stack_lis, info_stack_lis, self.beta)
        bin_lis, bin_lis_index, info_bin_lis, gamma = self.stripe_into_bin(stripe_lis, info_stripe_lis, self.gamma)
        file = self.picture(data, stack_lis, info_stack_lis, stripe_lis, info_stripe_lis, bin_lis)


        min_number = 0
        for index, i in enumerate(gamma):
            min_number += i[index]

        return min_number, file

    def item_into_stack(self, data, alpha):

        # 构造出 stack stak:[list1, list2,....],list 的第一个元素为 stack 的计算标号
        stack_lis = []
        for item_index, item in enumerate(data):
            if len(stack_lis) == 0:
```

```
                    stack_lis.append([item_index+1])
                else:

                    for index, i in enumerate(stack_lis):
                        if data[i[0] - 1][5] == item[5]:
                            stack_length = 0
                            for j in i:
                                stack_length += data[j-1][4]
                            if stack_length + item[4] <= self.average_length:
                                stack_lis[index].append(item_index+1)
                            else:
                                # 转换 h 和 w
                                # if data[i[0] - 1][5] == item[4]:
                                #             data[item_index][4],  data[item_index][5]  =
data[item_index][5], data[item_index][4]
                                #                     if  stack_length  +  item[4]  <=
self.average_length:
                                #                 stack_lis[index].append(item_index + 1)
                                # else:
                                #         stack_lis.append([item_index+1])
                                stack_lis.append([item_index+1])
                                break
                        else:
                            continue
                    sum = 0

                    for i in stack_lis:
                        for j in i:
                            if item_index + 1 == j:
                                sum += 1
                    if sum == 0:
                        stack_lis.append([item_index + 1])

            # 处理 alpha
            for vector in stack_lis:
                # 计算序号变为 python 序号

                set_vector = []
                for number in vector:
                    set_vector.append(number - 1)
                cycle_set = list(set(range(self.data_size)) - set(set_vector))
                for j in cycle_set:
                    alpha[vector[0] - 1][j] = 0
                if len(vector) > 1:
                    for j in vector[1:]:
                        for k in range(self.data_size):
                            alpha[j - 1][k] = 0

            # 输出 stack 的高度和宽度信息（先高后宽）
```

```
        info_stack = [[],[]]
        for i in stack_lis:
            stack_length = 0
            for j in i:
                stack_length += data[j - 1][4]
            info_stack[0].append(stack_length)
            info_stack[1].append(data[i[0] - 1][5])

        return stack_lis, info_stack, alpha

    def stack_into_stripe(self, stack, info_stack, beta):
        # 先对 stack 做一次排序
        stack_length = info_stack[0]
        stack_width = info_stack[1]
        sort_stack = []
        for index, i in enumerate(stack_width):
            sort_stack.append([stack_length[index], i, stack[index]])

        # 排序
        n = len(sort_stack)
        for i in range(n):
            min = i
            for j in range(i+1, n):
                if sort_stack[j][0] < sort_stack[min][0]:
                    min = j
            sort_stack[min], sort_stack[i] = sort_stack[i], sort_stack[min]
        sort_stack = sort_stack[::-1]

        # 分出数据
        finish_sort_stack = []
        sort_info_stack = []
        for i in sort_stack:
            sort_info_stack.append([i[0], i[1]])
            finish_sort_stack.append(i[2])

        # 构造出 stripe, stripe：[[number,element1, element2, ....], .....],number 为其
序号,element 为包含的 stack 序号
        stripe_lis = []
        for item_index, item in enumerate(finish_sort_stack):
            if len(stripe_lis) == 0:
                stripe_lis.append([item_index])
            else:
                for i_index, i in enumerate(stripe_lis):
                    stripe_width = 0
                    for j in i:
                        stripe_width += sort_info_stack[j][1]
                    if stripe_width + sort_info_stack[item_index][1] <=
self.max_width:
                        stripe_lis[i_index].append(item_index)
```

```
                            break
                        else:
                            continue
                    sum = 0
                    for i in stripe_lis:
                        for j in i:
                            if item_index == j:
                                sum += 1
                    if sum == 0:
                        stripe_lis.append([item_index])

        # 输出 stripe 的宽度信息
        info_stripe = []
        max_length_index = []
        for v_index, vector in enumerate(stripe_lis):
            stripe_width = 0
            stripe_length = []
            for j_index, j in enumerate(vector):
                stripe_length.append(sort_info_stack[j][0])
                stripe_width += sort_info_stack[j][1]
            info_stripe.append([max(stripe_length), stripe_width])
            max_length_index.append(stripe_length.index(max(stripe_length)))
        # 将 stripe 做序号转变
        true_stripe_lis = []
        for v_index, vector in enumerate(stripe_lis):
            max_index = max_length_index[v_index]
            true_stripe_lis.append([finish_sort_stack[vector[max_index]][0]])
            for j_index, j in enumerate(vector):
                true_stripe_lis[v_index].append(finish_sort_stack[j][0])

        # 处理 beta
        first_vector = []
        for vector in true_stripe_lis:
            # 计算序号变为 python 序号
            first_vector.append(vector[0]-1)
            set_vector = []
            for number in vector[1:]:
                set_vector.append(number - 1)
            cycle_set = list(set(range(self.data_size)) - set(set_vector))
            for j in cycle_set:
                beta[vector[0] - 1][j] = 0
        zero_lis = list(set(range(self.data_size)) - set(first_vector))
        for i in zero_lis:
            for j in range(self.data_size):
                beta[i][j] = 0

        return true_stripe_lis, stripe_lis, info_stripe, beta

    def stripe_into_bin(self, stripe, info_stripe, gamma):
```

```
        # 先对 stack 做一次排序
        sort_stripe = []
        for index, i in enumerate(stripe):
            sort_stripe.append([info_stripe[index][0], info_stripe[index][1], i])

        # 排序
        n = len(sort_stripe)
        for i in range(n):
            min = i
            for j in range(i + 1, n):
                if sort_stripe[j][0] < sort_stripe[min][0]:
                    min = j
            sort_stripe[min], sort_stripe[i] = sort_stripe[i], sort_stripe[min]
        sort_stripe = sort_stripe[::-1]

        # 分出数据
        finish_sort_stripe = []
        sort_info_stripe = []
        for i in sort_stripe:
            sort_info_stripe.append([i[0], i[1]])
            finish_sort_stripe.append(i[2])

        # 构造出 bin, bin：[[number,element1, element2, ....], .....],number 为其序号,element 为包含的 bin 序号
        bin_lis = []
        for item_index, item in enumerate(finish_sort_stripe):
            if len(bin_lis) == 0:
                bin_lis.append([item_index])
            else:
                for i_index, i in enumerate(bin_lis):
                    bin_length = 0
                    for j in i:
                        bin_length += sort_info_stripe[j][0]
                    if bin_length + sort_info_stripe[item_index][0] <= self.max_length:
                        bin_lis[i_index].append(item_index)
                        break
                    else:
                        continue
                sum = 0
                for i in bin_lis:
                    for j in i:
                        if item_index == j:
                            sum += 1
                if sum == 0:
                    bin_lis.append([item_index])

        # 输出 bin 的高宽信息
        info_bin = []
```

```
            max_length_index = []
            for v_index, vector in enumerate(bin_lis):
                bin_width = []
                bin_length = 0
                for j_index, j in enumerate(vector):
                    bin_width.append(sort_info_stripe[j][1])
                    bin_length += sort_info_stripe[j][0]
                info_bin.append([bin_length, max(bin_width)])
                max_length_index.append(bin_width.index(max(bin_width)))

            # 将 bin 做序号转变
            true_bin_lis = []
            for v_index, vector in enumerate(bin_lis):

true_bin_lis.append([finish_sort_stripe[vector[max_length_index[v_index]]][0]])
                for j_index, j in enumerate(vector):
                    true_bin_lis[v_index].append(finish_sort_stripe[j][0])

            # 处理 gamma
            first_vector = []
            for vector in true_bin_lis:
                # 计算序号变为 python 序号
                first_vector.append(vector[0] - 1)
                set_vector = []
                for number in vector[1:]:
                    set_vector.append(number - 1)
                cycle_set = list(set(range(self.data_size)) - set(set_vector))
                for j in cycle_set:
                    gamma[vector[0] - 1][j] = 0
            zero_lis = list(set(range(self.data_size)) - set(first_vector))
            for i in zero_lis:
                for j in range(self.data_size):
                    gamma[i][j] = 0

            return true_bin_lis, bin_lis, info_bin, gamma

        def picture(self, data, stack_lis, info_stack_lis, stripe_lis, info_stripe_lis, bin_lis):
            # 找出每个 bin 所含的订单 id
            # 制作 bin、stripe、stack 的高宽信息字典
            info_stack_dict = {}
            for stack_index, stack_i in enumerate(stack_lis):
                info_stack_dict[stack_i[0]] = [info_stack_lis[0][stack_index],
info_stack_lis[1][stack_index]]
            info_stripe_dict = {}
            for stripe_index, stripe_i in enumerate(stripe_lis):
                info_stripe_dict[stripe_i[0]] = info_stripe_lis[stripe_index]

            stripe_dict = {}
            for v_index, vector in enumerate(stripe_lis):
```

```
                stripe_dict[vector[0]] = vector[1:]

        stack_dict = {}
        for v_index, vector in enumerate(stack_lis):
            stack_dict[vector[0]] = vector

        # 整合为 all_bin_lis
        all_bin_lis = []
        for v_index, vector in enumerate(bin_lis):
            all_bin_lis.append([vector[0]])
            for i_index, index_1 in enumerate(vector[1:]):
                all_bin_lis[-1].append([index_1, stripe_dict.get(index_1)])

        for b_index, bin in enumerate(all_bin_lis):
            for sp_index, stripe in enumerate(bin[1:]):
                for sc_index, stack in enumerate(stripe[1:]):
                    for it_index, item in enumerate(stack):
                        all_bin_lis[b_index][sp_index + 1][sc_index + 1][it_index]
= [item, stack_dict.get(item)]

        # 制作 cut_program.csv 表
        # 小心计算序号是 1 到 n,引用序号要变成 0 到 n-1
        # [材质, bin_index, item_index, 起点 x 坐标, 起点 y 坐标, x 方向长度(width),
y 方向长度(length)]
        cut_program_lis = []
        for bin_index, bin in enumerate(all_bin_lis):
            # 整个图
            fig, ax = plt.subplots()
            ax.spines['right'].set_visible(False)
            ax.spines['top'].set_visible(False)
            ax.spines['left'].set_visible(False)
            ax.spines['bottom'].set_visible(False)
            ax.plot([0, 1220], [0, 0], color="k", linewidth=2.5)
            ax.plot([1220, 1220], [0, 2440], color="k", linewidth=2.5)
            ax.plot([0, 0], [0, 2440], color="k", linewidth=2.5)
            ax.plot([0, 1220], [2440, 2440], color="k", linewidth=2.5)

            # stripe 的直线位置
            total_stripe_length = 0.
            for stripe_index, stripe in enumerate(bin[1:]):
                stripe_length, stripe_width = info_stripe_dict.get(stripe[0])[0],
info_stripe_dict.get(stripe[0])[1]
                total_stripe_length += stripe_length
                ax.plot([0, 1220], [total_stripe_length, total_stripe_length], color="k",
linewidth=2)

                # stack 的竖线位置
                total_stack_width = 0.
                for stack_index, stack in enumerate(*stripe[1:]):
```

```
                stack_length, stack_width = info_stack_dict.get(stack[0])[0], info_stack_dict.get(stack[0])[1]
                total_stack_width += stack_width
                ax.plot([total_stack_width, total_stack_width], [total_stripe_length - stripe_length, total_stripe_length], color="g", linewidth=1.5)

                # item 的位置
            total_item_length = 0.
            for item_index, item in enumerate(*stack[1:]):
                item_length, item_width = data[item - 1][4], data[item - 1][5]
                total_item_length += item_length

                cut_program_lis.append([data[item - 1][2], bin[0], data[item - 1][1], total_stack_width - stack_width, total_item_length - item_length + total_stripe_length - stripe_length, stack_width, item_length])
                ax.plot([total_stack_width - stack_width, total_stack_width], [total_item_length + total_stripe_length - stripe_length, total_item_length+ total_stripe_length - stripe_length], color="y", linewidth=1)
                adjust_text([ax.text(x=(total_stack_width - stack_width + total_stack_width) / 2, y= (total_item_length - 3*item_length/10) + total_stripe_length - stripe_length , rotation=60, s="id:{0}".format(data[item - 1][1]), ha="left")])
            ax.fill_between([total_stack_width - stack_width, total_stack_width], total_item_length + total_stripe_length - stripe_length, stripe_length + total_stripe_length - stripe_length, facecolor='dimgray', alpha=1)
        ax.fill_between([total_stack_width, 1220], total_stripe_length - stripe_length, total_stripe_length, facecolor='dimgray', alpha=1)
    ax.fill_between([0, 1220], total_stripe_length, 2440, facecolor='dimgray', alpha=1)
    paths = os.path.join(self.path, "cutting_pic{0}.jpg".format(bin[0]))
    plt.savefig(paths)
    plt.close()


    cut_program = pd.DataFrame(cut_program_lis)
cut_program.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']
return cut_program
```

附录 3

介绍：问题二的主代码

```
def metrics(data, number, max_length, max_width):
    """
    :parameter
    data:[[序号：index, 订单 id:item_id, 订单材料:item_material, 订单所需要数量:item_num, 材料长度:item_length, 材料宽度:item_width, 订单号:item_order]*size]
    number:消耗的总板材
    max_length:板材长
    max_width:板材宽
```

```
    """
    total_item = 0.
    for vector in data:
        total_item += vector[4] * vector[5]
    run_rate = total_item / (number*max_length*max_width)

    return run_rate

data_B1_material_dict = {'NJYH-0225S': 0, 'PKQL-0418SD': 1, 'YHXM-0218S': 2, 'FMB-0215S': 3, 'MNH-0225SD': 4, 'SRBW-0218SD': 5, 'NJYH-0215S': 6, 'FMB-0235S': 7, 'CBZP-0218S': 8, 'LMW-0218SD': 9, 'SL-0215S': 10, 'PKQL-0418S': 11, 'YW8-0218SD': 12, 'HYBW-0218S': 13, 'ZYF-0218S': 14, 'QKQ-0218SD': 15, 'QKQ-0215S': 16, 'NHQL-0218S': 17, 'GHSS-0218S': 18, 'SRBW-0218S': 19, 'YW1-0225S': 20, 'ZQB-0215S': 21, 'YW1-0218SD': 22, 'XMJQ-0225S': 23, 'JH-0218S': 24, 'XMJQ-0218S': 25, 'XXB-0218SD': 26, 'NJBB-0215S': 27, 'XMJQ-0218SD': 28, 'XYHX-0225S': 29, 'YGKQ-0418SD': 30, 'MNH-0215S': 31, 'XYH-0215S': 32, 'YHXM-0215S': 33, 'NMYH-0218S': 34, 'XXB-0225S': 35, 'YDBS-0218S': 36, 'NBSY-0235S': 37, 'LMW-0225S': 38, 'NHJB-0218S': 39, 'ALJQ-0218S': 40, 'SL-0218S': 41, 'YW1-0215S': 42, 'PGBL-0418SD': 43, 'HYBW-0225S': 44, '5-0215S': 45, 'YDBS-0225S': 46, 'PGBL-0418S': 47, 'NMYH-0215S': 48, 'NBSY-0218SD': 49, '5-0218S': 50, '5-0218SD': 51, 'YW8-0218S': 52, 'NJHL-0215S': 53, 'FMB-0218S': 54, 'YW1-0218S': 55, 'JH-0215S': 56, 'ZYF-0215S': 57, 'ZZL-0218S': 58, 'YW10-0225S': 59, 'XXB-0218S': 60, 'QKQ-0218S': 61, 'MNH-0218S': 62, 'HT9-0225S': 63, 'MNH-0225S': 64, 'NJBB-0218S': 65, 'XYHX-0218S': 66, 'NBSY-0218S': 67, 'NBSY-0225S': 68, 'MNH-0218SD': 69, 'GDMY-0218S': 70, 'ALJQ-0215S': 71, 'LMW-0215S': 72, 'XXB-0215S': 73, 'LSXBJ-0418S': 74, 'XMJQ-0215S': 75, 'GDMY-0218SD': 76, 'HT9-0218S': 77, 'ZYF-0218SD': 78, 'GDMY-0215S': 79, 'YTM-0225S': 80, 'ALJQ-0218SD': 81, 'GGXK-0418SD': 82, 'YGFX-0218S': 83, 'LMW-0218S': 84, 'ZQB-0218S': 85, 'YW1-0235S': 86, 'PDLL-0418S': 87, 'FMB-0225S': 88, 'GHSS-0218S-D': 89, 'ZQB-0225S': 90, 'GDMY-0225SD': 91, 'YTM-0218S': 92, 'YW10-0215S': 93, 'QSYM-0225S': 94, 'YMH-0215S': 95, 'YDBS-0215S': 96, 'YMH-0218S': 97, 'QSYM-0218S': 98, 'GHSS-0215S': 99, 'YGKQ-0418S': 100, 'YW10-0218S': 101, 'NJHL-0218S': 102, 'NJYH-0218SD': 103, 'YW8-0215S': 104, 'YMH-0225S': 105, 'NDSF-0218S': 106, 'YTM-0215S': 107, 'XYHX-0215S': 108, 'QKQ-0225S': 109, 'GDMY-0225S': 110, 'YSH-0215S': 111, 'PSH-0418S': 112, 'YSH-0218SD': 113, 'NBSY-0215S': 114, 'NJYH-0218S': 115, 'HYBW-0215S': 116, 'NHJB-0215S': 117, 'GGXK-0418S': 118, 'CGS-0218SD': 119, 'HT9-0215S': 120, 'SRBW-0215S': 121, 'YDBS-0218SD': 122, 'QSYM-0215S': 123, 'CGS-0218S': 124, 'FMB-0218SD': 125, 'NBSY-0225SD': 126, 'YMH-0218SD': 127, 'YSH-0218S': 128, 'PSHL-0418S': 129}

data_B2_material_dict = {'LTNS-0225S': 0, 'LTNS-0215S': 1, 'NJYH-0225S': 2, 'PKQL-0418SD': 3, 'YHXM-0218S': 4, '084-0218S': 5, 'FMB-0215S': 6, 'MNH-0225SD': 7, 'SRBW-0218SD': 8, 'NJYH-0215S': 9, 'FMB-0235S': 10, 'LMW-0218SD': 11, 'CBZP-0218S': 12, 'SL-0215S': 13, 'YGFX-0225S': 14, 'PKQL-0418S': 15, 'QSYM-0218SD': 16, 'HYBW-0218S': 17, 'ZYF-0218S': 18, 'CBZP-0215S': 19, 'PLHH-0418S': 20, 'QKQ-0218SD': 21, 'QKQ-0215S': 22, 'GHSS-0218S': 23, '5-0225S': 24, 'SRBW-0218S': 25, 'ZYF-0225S': 26, 'YW1-0225S': 27, 'ZQB-0215S': 28, 'PSHL-0418S': 29, 'XMJQ-0225S': 30, 'JH-0218S': 31, 'YDH-0215S': 32, 'XMJQ-0218S': 33, 'PAM-0225S': 34, 'LTNS-0218S': 35, 'XXB-0218SD': 36, 'PAM-0215S': 37, 'XMJQ-0218SD': 38, 'PSXH-0418S': 39, 'YGKQ-0418SD': 40, 'MNH-0215S': 41, 'ZQB-0225SD': 42, 'NMYH-0218S': 43, 'XXB-0225S': 44, 'YDBS-0218S': 45, 'NBSY-0235S': 46, 'LMW-0225S': 47, 'PAM-
```

0218SD': 48, 'CGYM-0218S': 49, 'SL-0218S': 50, 'PGBL-0418SD': 51, 'YW1-0215S': 52, 'YH-0218S': 53, '084-0215S': 54, 'HYBW-0225S': 55, '5-0215S': 56, 'PGBL-0418S': 57, 'YDBS-0225S': 58, 'NMYH-0215S': 59, 'YSH-0225S': 60, 'XYHX-0218SD': 61, '5-0218S': 62, 'YH-0215S': 63, 'NSLH-0218S': 64, 'NJHL-0215S': 65, 'FMB-0218S': 66, 'YW1-0218S': 67, 'JH-0215S': 68, 'ZYF-0215S': 69, 'ZZL-0218S': 70, 'PSHL-0418SD': 71, 'XXB-0218S': 72, 'QKQ-0218S': 73, 'HT9-0225S': 74, 'MNH-0218S': 75, 'FMS-0215S': 76, 'MNH-0225S': 77, 'PAM-0218S': 78, 'XYHX-0218S': 79, 'YGFX-0215S': 80, 'HT9-0235S': 81, 'NBSY-0218S': 82, 'PHJS-0418SD': 83, 'NBSY-0225S': 84, 'MNH-0218SD': 85, 'GDMY-0218S': 86, 'GHSS-0225S': 87, 'FMS-0225S': 88, 'CGYM-0215S': 89, 'CGYM-0225S': 90, 'ZZL-0215S': 91, 'LMW-0215S': 92, 'XXB-0215S': 93, 'XMJQ-0215S': 94, 'GDMY-0218SD': 95, 'YDH-0225S': 96, 'YDH-0218S': 97, 'HT9-0218S': 98, 'SL-0225S': 99, 'GDMY-0215S': 100, 'LTNS-0218SD': 101, 'YGFX-0218S': 102, 'LMW-0218S': 103, 'YSH-0218S': 104, 'ZQB-0218S': 105, 'PDLL-0418S': 106, 'FMB-0225S': 107, 'FMS-0218SD': 108, 'YSH-0225SD': 109, 'ZQB-0225S': 110, 'PHYB-0418S': 111, 'YDH-0218SD': 112, 'QSYM-0225S': 113, 'YMH-0215S': 114, 'YDBS-0215S': 115, 'YMH-0218S': 116, 'PSXH-0418SD': 117, 'QSYM-0218S': 118, 'GHSS-0215S': 119, 'YGKQ-0418S': 120, 'YW10-0218S': 121, 'NJHL-0218S': 122, 'NJYH-0218SD': 123, 'PSKQL-0418S': 124, 'NDSF-0218S': 125, 'YSH-0215S': 126, 'PSH-0418S': 127, 'YSH-0218SD': 128, 'NBSY-0215S': 129, 'PTJH-0418S': 130, 'XXB-0235S': 131, 'NJYH-0218S': 132, 'HYBW-0215S': 133, 'HT9-0215S': 134, 'PBWS-0418S': 135, 'FMS-0218S': 136, 'QSYM-0215S': 137, 'PHJS-0418S': 138, 'ZZL-0225S': 139, 'FMB-0218SD': 140, 'NSLH-0215S': 141, 'YMH-0218SD': 142, 'XYHX-0215S': 143, '5-0218SD': 144, 'YGFX-0218SD': 145}

data_B3_material_dict = {'LTNS-0215S': 0, 'JH-0225S': 1, 'NDSF-0215S': 2, 'YHXM-0218S': 3, '084-0218S': 4, 'NMYH-0225S': 5, 'FMB-0215S': 6, 'SRBW-0218SD': 7, 'CBZP-0218S': 8, 'NHQL-0215S': 9, 'SL-0215S': 10, 'YGFX-0225S': 11, 'PKQL-0418S': 12, 'QSYM-0218SD': 13, 'NDSF-0225S': 14, 'YW8-0218SD': 15, 'HYBW-0218S': 16, 'ZYF-0218S': 17, 'CBZP-0215S': 18, 'NJHL-0218SD': 19, 'QKQ-0218SD': 20, 'QKQ-0215S': 21, 'NHQL-0218S': 22, 'GHSS-0218S': 23, '5-0225S': 24, 'SRBW-0218S': 25, 'ZYF-0225S': 26, 'YW1-0225S': 27, 'ZQB-0215S': 28, 'YW1-0218SD': 29, 'CGS-0215S': 30, 'PSHL-0418S': 31, 'XMJQ-0225S': 32, 'JH-0218S': 33, 'XMJQ-0218S': 34, 'YDH-0215S': 35, 'LTNS-0218S': 36, 'XXB-0218SD': 37, 'NHLP-0218S': 38, 'XMJQ-0218SD': 39, 'YH-0225S': 40, 'PSXH-0418S': 41, 'XYHX-0225S': 42, 'YGKQ-0418SD': 43, 'MNH-0215S': 44, 'YHXM-0215S': 45, 'NMYH-0218S': 46, 'XXB-0225S': 47, 'YDBS-0218S': 48, '084-0225S': 49, 'CGYM-0218S': 50, 'KLXQ-0218SD': 51, 'NHJB-0218S': 52, 'SL-0218S': 53, 'YW1-0215S': 54, 'YH-0218S': 55, '084-0215S': 56, 'HYBW-0225S': 57, '5-0215S': 58, 'PGBL-0418S': 59, 'YDBS-0225S': 60, 'NMYH-0215S': 61, 'NBSY-0218SD': 62, 'KLXQ-0215S': 63, 'KLXQ-0225SD': 64, 'XYHX-0225SD': 65, 'XYHX-0218SD': 66, '5-0218S': 67, 'YDH-0235S': 68, 'YH-0215S': 69, 'YW8-0218S': 70, 'NSLH-0218S': 71, 'FMB-0218S': 72, 'YW1-0218S': 73, 'JH-0215S': 74, 'ZYF-0215S': 75, 'ZZL-0218S': 76, 'YW10-0225S': 77, 'XXB-0218S': 78, 'YHXM-0225S': 79, 'QKQ-0225SD': 80, 'QKQ-0218S': 81, 'MNH-0218S': 82, 'HT9-0225S': 83, 'MNH-0225S': 84, 'NJBB-0218S': 85, 'PAM-0218S': 86, 'XYHX-0218S': 87, 'YGFX-0215S': 88, 'YHXM-0218SD': 89, 'NBSY-0218S': 90, 'NBSY-0225S': 91, '5-0225SD': 92, 'MNH-0218SD': 93, 'GDMY-0218S': 94, 'CGYM-0215S': 95, 'LMW-0215S': 96, 'XXB-0215S': 97, 'LSXBJ-0418S': 98, 'XMJQ-0215S': 99, 'GDMY-0218SD': 100, 'SRBW-0225S': 101, 'YDH-0225S': 102, 'YDH-0218S': 103, 'CGYM-0235S': 104, 'SL-0225S': 105, 'GDMY-0215S': 106, 'ZYF-0218SD': 107, 'LTNS-0218SD': 108, 'HT9-0218S': 109, 'YGFX-0218S': 110, 'GGXK-0418SD': 111, 'LMW-0218S': 112, 'YSH-0218S': 113, 'YGFX-0225SD': 114, 'ZQB-0218S': 115, 'PDLL-

0418S': 116, 'FMB-0225S': 117, 'GHSS-0218S-D': 118, 'ZQB-0225S': 119, 'GDMY-0225SD': 120, 'YDH-0218SD': 121, 'YW10-0215S': 122, 'QSYM-0225S': 123, 'YMH-0215S': 124, 'YDBS-0215S': 125, 'LTNS-0225SD': 126, 'YMH-0218S': 127, 'PSXH-0418SD': 128, 'QSYM-0218S': 129, 'YGKQ-0418S': 130, 'YW10-0218S': 131, 'NJHL-0218S': 132, 'YW8-0215S': 133, 'YGF-0218S': 134, 'NDSF-0218S': 135, 'QKQ-0225S': 136, 'GDMY-0225S': 137, 'YSH-0215S': 138, 'PSH-0418S': 139, 'KLXQ-0218S': 140, 'NBSY-0215S': 141, 'XXB-0235S': 142, 'GHSS-0225SD': 143, 'NJYH-0218S': 144, 'HYBW-0215S': 145, 'NHJB-0215S': 146, 'GGXK-0418S': 147, 'CGS-0218SD': 148, 'HT9-0215S': 149, 'SRBW-0215S': 150, 'YDBS-0218SD': 151, 'QSYM-0215S': 152, 'CGS-0218S': 153, 'NBSY-0225SD': 154, 'NSLH-0215S': 155, 'YMH-0218SD': 156, 'XYHX-0215S': 157, '5-0218SD': 158, 'YGFX-0218SD': 159}

data_B4_material_dict = {'NJYH-0225S': 0, 'JH-0225S': 1, 'NDSF-0215S': 2, 'PKQL-0418SD': 3, '084-0218S': 4, 'YHXM-0218S': 5, 'NMYH-0225S': 6, 'FMB-0215S': 7, 'MNH-0225SD': 8, 'NJYH-0215S': 9, 'FMB-0235S': 10, 'CBZP-0218S': 11, 'LMW-0218SD': 12, 'SL-0215S': 13, 'PKQL-0418S': 14, 'NDSF-0225S': 15, 'YMH-0225SD': 16, 'HYBW-0218S': 17, 'CBZP-0215S': 18, 'ZYF-0218S': 19, 'NJHL-0218SD': 20, 'YW1-0225SD': 21, 'XMJQ-0225SD': 22, 'QKQ-0218SD': 23, 'QKQ-0215S': 24, 'GHSS-0218S': 25, 'PDLL-0418SD': 26, 'ZQB-0218SD': 27, 'SRBW-0218S': 28, 'YW1-0225S': 29, 'ZQB-0215S': 30, 'YW1-0218SD': 31, 'XMJQ-0225S': 32, 'JH-0218S': 33, 'XMJQ-0218S': 34, 'XXB-0218SD': 35, 'PAM-0215S': 36, 'NHLP-0218S': 37, 'XMJQ-0218SD': 38, 'NHLP-0215S': 39, 'PSXH-0418S': 40, 'YH-0225S': 41, 'YGKQ-0418SD': 42, 'MNH-0215S': 43, 'YHXM-0215S': 44, 'NMYH-0218S': 45, 'XXB-0225S': 46, 'YDBS-0218S': 47, '084-0225S': 48, 'NBSY-0235S': 49, 'LMW-0225S': 50, 'PAM-0218SD': 51, 'PGBL-0418SD': 52, 'YW1-0215S': 53, 'SL-0218S': 54, 'HYBW-0225S': 55, '084-0215S': 56, 'YH-0218S': 57, '5-0215S': 58, 'YDBS-0225S': 59, 'PGBL-0418S': 60, 'NBSY-0218SD': 61, 'NMYH-0215S': 62, 'QKQ-0235S': 63, 'XYHX-0218SD': 64, '5-0218S': 65, '5-0218SD': 66, 'YH-0215S': 67, 'NJHL-0215S': 68, 'FMB-0218S': 69, 'YW1-0218S': 70, 'JH-0215S': 71, 'ZYF-0215S': 72, 'ZZL-0218S': 73, 'PSHL-0418SD': 74, 'XXB-0218S': 75, 'YTM-0235S': 76, 'YHXM-0225S': 77, 'PBWS-0418SD': 78, 'QKQ-0218S': 79, 'MNH-0218S': 80, 'MNH-0225S': 81, 'PAM-0218S': 82, 'XYHX-0218S': 83, 'YHXM-0218SD': 84, 'NBSY-0218S': 85, 'NBSY-0225S': 86, 'MNH-0218SD': 87, 'GDMY-0218S': 88, 'GHSS-0225S': 89, '084-0218SD': 90, 'ZZL-0215S': 91, 'LMW-0215S': 92, 'XXB-0215S': 93, 'LSXBJ-0418S': 94, 'XMJQ-0215S': 95, 'GDMY-0218SD': 96, 'HT9-0218S': 97, 'GDMY-0215S': 98, 'ZYF-0218SD': 99, 'YTM-0225S': 100, 'LMW-0218S': 101, 'ZQB-0218S': 102, 'PDLL-0418S': 103, 'YW1-0235S': 104, 'YTM-0218S': 105, 'PHYB-0418S': 106, 'YMH-0215S': 107, 'YDBS-0215S': 108, 'YMH-0218S': 109, 'PSXH-0418SD': 110, 'QSYM-0218S': 111, 'GHSS-0215S': 112, 'YGKQ-0418S': 113, 'NJHL-0218S': 114, 'NJYH-0218SD': 115, 'YMH-0225S': 116, 'NDSF-0218S': 117, 'YTM-0215S': 118, '084-0225SD': 119, 'XYHX-0215S': 120, 'QKQ-0225S': 121, 'YSH-0215S': 122, 'PSH-0418S': 123, 'YSH-0218SD': 124, 'NBSY-0215S': 125, 'PTJH-0418S': 126, 'XXB-0235S': 127, 'NJYH-0218S': 128, 'NHJB-0215S': 129, 'HYBW-0215S': 130, 'GGXK-0418S': 131, 'PBWS-0418S': 132, 'SRBW-0215S': 133, 'YDBS-0218SD': 134, 'QSYM-0215S': 135, 'CGS-0218S': 136, 'NSLH-0235S': 137, 'NSLH-0215S': 138, 'YMH-0218SD': 139, 'YSH-0218S': 140, 'PSHL-0418S': 141}

data_B5_material_dict = {'LTNS-0225S': 0, 'NJYH-0225S': 1, 'LTNS-0215S': 2, 'NDSF-0215S': 3, 'PKQL-0418SD': 4, 'YHXM-0218S': 5, 'NMYH-0225S': 6, 'FMB-0215S': 7, 'ZHM-0215S': 8, 'SRBW-0218SD': 9, 'XYH-0218SD': 10, 'NJYH-0215S': 11, 'CBZP-0218S': 12, 'NHQL-0215S': 13, 'LMW-0218SD': 14, 'SL-0215S': 15, 'FMB-0225SD': 16, 'PKQL-0418S': 17, 'YGFX-0225S': 18, 'NDSF-0225S': 19, 'HYBW-0218S': 20, 'ZYF-0218S': 21, 'CBZP-0215S': 22, 'NJHL-0218SD': 23, 'YW1-0225SD': 24, 'QKQ-0218SD':

```
25, 'QKQ-0215S': 26, 'NHQL-0218S': 27, 'GHSS-0218S': 28, '5-0225S': 29, 'ZQB-0218SD': 30, 'SRBW-0218S': 31, 'ZYF-0225S': 32, 'YGF-0218SD': 33, 'ZQB-0215S': 34, 'YW1-0225S': 35, 'YW1-0218SD': 36, 'PSHL-0418S': 37, 'XMJQ-0225S': 38, 'JH-0218S': 39, 'XMJQ-0218S': 40, 'YDH-0215S': 41, 'PAM-0225S': 42, 'LTNS-0218S': 43, 'CBZP-0225S': 44, 'XXB-0218SD': 45, 'NJBB-0215S': 46, 'PAM-0215S': 47, 'NJBB-0225S': 48, 'XMJQ-0218SD': 49, 'YH-0225S': 50, 'PSXH-0418S': 51, 'XYHX-0225S': 52, 'YGKQ-0418SD': 53, 'MNH-0215S': 54, 'XYH-0215S': 55, 'YHXM-0215S': 56, 'XXB-0225S': 57, 'NMYH-0218S': 58, 'YDBS-0218S': 59, 'NBSY-0235S': 60, 'LMW-0225S': 61, 'PAM-0218SD': 62, 'CGYM-0218S': 63, 'NHJB-0218S': 64, 'ALJQ-0218S': 65, 'NSLH-0225S': 66, 'ZHM-0218S': 67, 'NHQL-0225S': 68, 'SL-0218S': 69, 'PGBL-0418SD': 70, 'YW1-0215S': 71, 'HYBW-0225S': 72, 'YH-0218S': 73, '5-0215S': 74, 'YDBS-0225S': 75, 'PGBL-0418S': 76, 'NBSY-0218SD': 77, 'NMYH-0215S': 78, 'YSH-0225S': 79, 'XYHX-0225SD': 80, 'XYHX-0218SD': 81, '5-0218S': 82, 'YW8-0218S': 83, 'YH-0215S': 84, 'NSLH-0218S': 85, 'NJHL-0215S': 86, 'FMB-0218S': 87, 'YW1-0218S': 88, 'JH-0215S': 89, 'ZYF-0215S': 90, 'ZZL-0218S': 91, 'YW10-0225S': 92, 'PSHL-0418SD': 93, 'XXB-0218S': 94, 'YHXM-0225S': 95, 'YTM-0235S': 96, 'PBWS-0418SD': 97, 'QKQ-0225SD': 98, 'QKQ-0218S': 99, 'MNH-0218S': 100, 'HT9-0225S': 101, 'FMS-0215S': 102, 'MNH-0225S': 103, 'NJBB-0218S': 104, 'PAM-0218S': 105, 'XYHX-0218S': 106, 'YGFX-0215S': 107, 'YHXM-0218SD': 108, 'NBSY-0218S': 109, 'NBSY-0225S': 110, 'MNH-0218SD': 111, 'NJYH-0225SD': 112, 'GDMY-0218S': 113, 'GHSS-0225S': 114, 'ALJQ-0215S': 115, 'XYH-0218S': 116, 'CGYM-0215S': 117, 'CGYM-0225S': 118, 'FMS-0225S': 119, 'ZZL-0215S': 120, 'LMW-0215S': 121, 'XXB-0215S': 122, 'XMJQ-0215S': 123, 'GDMY-0218SD': 124, 'YDH-0218S': 125, 'HT9-0218S': 126, 'SL-0225S': 127, 'GDMY-0215S': 128, 'ZYF-0218SD': 129, 'LTNS-0218SD': 130, 'YTM-0225S': 131, 'ALJQ-0218SD': 132, 'ZYF-0225SD': 133, 'YGFX-0218S': 134, 'LMW-0218S': 135, 'ZQB-0218S': 136, 'PDLL-0418S': 137, 'FMB-0225S': 138, 'YSH-0225SD': 139, 'GHSS-0218S-D': 140, 'ZQB-0225S': 141, 'YTM-0218S': 142, 'YW10-0215S': 143, 'QSYM-0225S': 144, 'YMH-0215S': 145, 'YDBS-0215S': 146, 'YMH-0218S': 147, 'PSXH-0418SD': 148, 'QSYM-0218S': 149, 'GHSS-0215S': 150, 'YGKQ-0418S': 151, 'YW10-0218S': 152, 'NJHL-0218S': 153, 'NJYH-0218SD': 154, 'YW8-0215S': 155, 'YGF-0218S': 156, 'NDSF-0218S': 157, 'YTM-0215S': 158, 'PSKQL-0418S': 159, 'XYH-0225S': 160, 'XYHX-0215S': 161, 'QKQ-0225S': 162, 'GDMY-0225S': 163, 'YSH-0215S': 164, 'PSH-0418S': 165, 'KLXQ-0218S': 166, 'YSH-0218SD': 167, 'NBSY-0215S': 168, 'NJYH-0218S': 169, 'NHJB-0215S': 170, 'HYBW-0215S': 171, 'GGXK-0418S': 172, 'CGS-0218SD': 173, 'HT9-0215S': 174, 'PBWS-0418S': 175, 'SRBW-0215S': 176, 'ZQB-0235S': 177, 'YDBS-0218SD': 178, 'FMS-0218S': 179, 'YW8-0225S': 180, 'CGS-0218S': 181, 'YGF-0215S': 182, 'ZZL-0225S': 183, 'FMB-0218SD': 184, 'XXB-0225SD': 185, 'NBSY-0225SD': 186, 'NSLH-0215S': 187, 'YMH-0218SD': 188, 'YSH-0218S': 189, '5-0218SD': 190, 'YGFX-0218SD': 191}

# file_A = []
# for i in range(1, 6):
#         file_path = os.path.join(r"E:\python document\challenge\2022 年 B 题\子问题 2-数据集 B", "new_data_B"+str(i)+".csv")
#         if os.path.exists(file_path):
#                 file_A.append(pd.read_csv(file_path))

file1 = pd.read_csv(r"E:\python document\challenge\2022 年 B 题\子问题 2-数据集 B\new_data_B4.csv")
```

```
file2 = pd.read_csv(r"E:\python document\challenge\2022 年 B 题\子问题 2-数据集 B\new_data_B5.csv")
# [序号：index,订单 id:item_id, 订单材料:item_material, 订单所需要数量:item_num, 材料长度:item_length, 材料宽度:item_width, 订单号:item_order,\
# mm 面积:item_area, dm 面积：item_area_10000, m 面积：item_area_1000000, 分类号:cluster_index]

total_data_set = []
for file in [file1, file2]:
    data_set = []
    cluster_lis = set(file["cluster_index"].values.tolist())
    for cluster_set_num in cluster_lis:
        cluster_groups = file.groupby(file.cluster_index)
        cluster_pd = cluster_groups.get_group(cluster_set_num)
        material_set = set(cluster_pd["item_material"].values.tolist())
        print(material_set)
        for mat in material_set:

            mat_groups = cluster_pd.groupby(cluster_pd.item_material)
            data_set.append(mat_groups.get_group(mat))
    total_data_set.append(data_set)
    # print(data_set)



# 参数
max_length = 2440
max_width = 1220
path = r"E:\python document\challenge\2022 年 B 题\solution_B"
f_txt = open(r"E:\python document\challenge\2022 年 B 题\solution_B\min_n.txt", "w")
min_n_data_B = []
rate_B_lis = []
run_time_lis = []
_time = time.time()
for file_index, file in enumerate(total_data_set):
    min_n_lis = []
    rate_lis = []
    for data_index, data in enumerate(file):
        para = data['item_length'].mean()
        j = data.sort_values(by='item_length', ascending=False)
        j = j.values.tolist()

        data_size = len(j)
        paths = os.path.join(path, "B{0}_{1}_{2}".format(file_index+4, j[0][10], j[0][2]))
        os.mkdir(paths)
        solution = Cycle_solution_A(data_size, paths, para, max_length, max_width)
        min_n, cut_program = solution.cycle_com(j)
```

```
            cut_program_1 = pd.DataFrame([j[0][10]]*data_size)
            cut_program = pd.concat([cut_program_1, cut_program], axis=1,
ignore_index=True)
            df = pd.DataFrame()
            df = pd.concat([cut_program, df], axis=0)
            run_time_lis.append(time.time() - _time)
            min_n_lis.append(min_n)
            rate = metrics(j, min_n, max_length, max_width)
            rate_lis.append(rate)

      total_min_n = 0
      for min_n in min_n_lis:
            total_min_n += total_min_n

      rate_size = len(rate_lis)
      rate_total = 0
      for rate in rate_lis:
            rate_total += rate
      total_rate = rate_total/rate_size

      min_n_data_B.append(total_min_n)
      rate_B_lis.append(total_rate)

      print(min_n_data_B)
      print(rate_B_lis)
      print(run_time_lis)

      df.to_csv(r"E:\python document\challenge\2022 年 B 题
\solution_B\sun_orderB{0}.csv".format(file_index + 4), index=None,
encoding='utf_8_sig')
      print("data_b{0}:finish".format(file_index + 4))
```

附录 4

介绍：问题二的 K-Means 聚类代码

```
b1 = pd.read_csv(r"E:\python document\challenge\2022 年 B 题\子问题 2-数据集
B\gaibian_data_B5.csv")
b1 = b1.sort_values(by='item_order', ascending=True)
order_set = set(b1['item_order'].values.tolist())
material_set = set(b1['item_material'].values.tolist())

b1 = b1.values.tolist()
data_size = len(b1)
max_cluster_size = math.ceil(data_size/1000)
data_lis = []
```

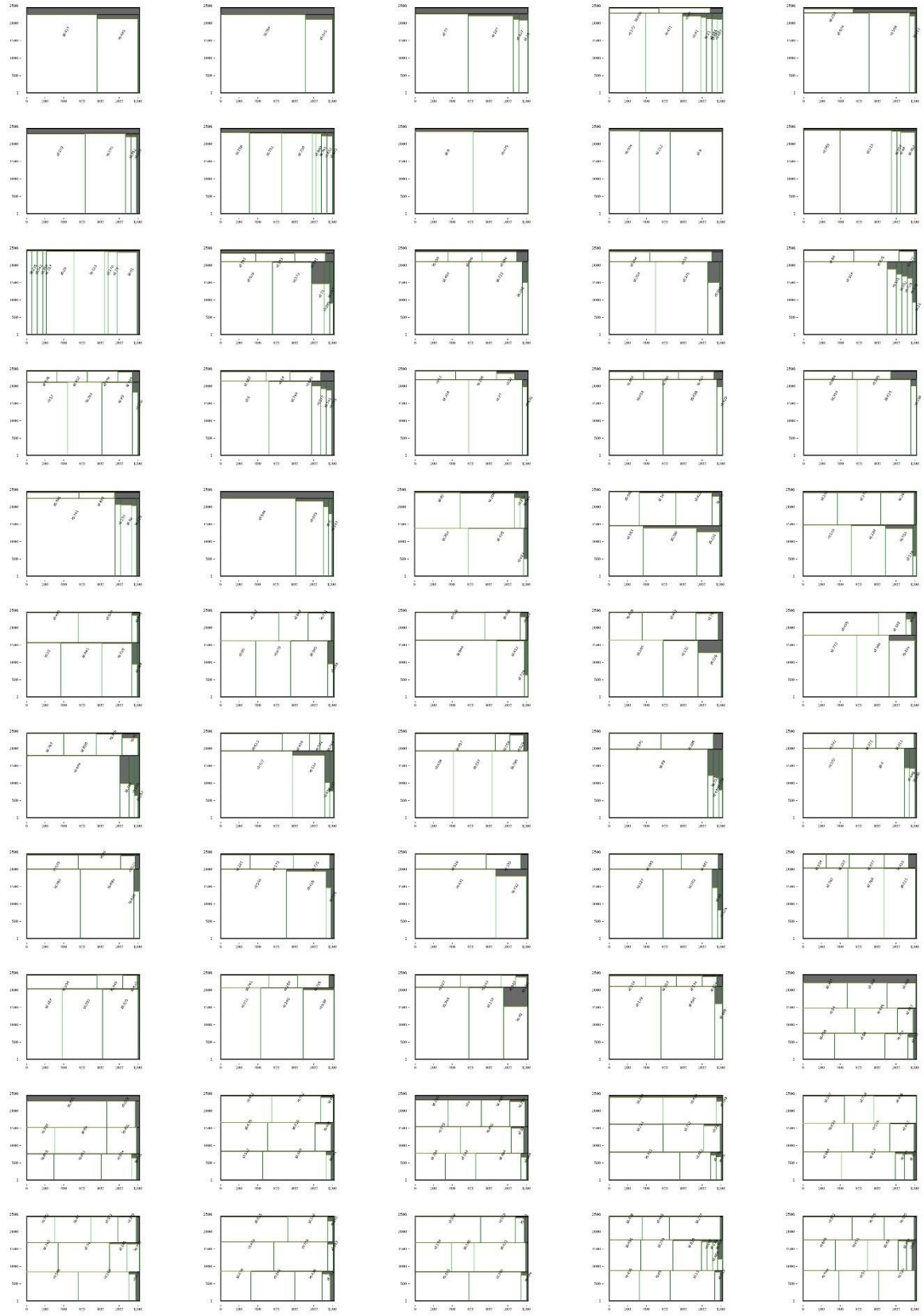
```
# 有两种形式，都试试
# 第一种 ai：[order, material, id, total_area, total_item_id] id 和 material 为 list
# 第二种 ai/b：[order, material/total_item_id, id, total_area, total_item_id] id 和 material 为 list
for order_num in order_set:
    data_lis.append([order_num, [], []])

for index, i in enumerate(data_lis):
    total_area = 0.
    total_item_id = 0
    for vector in b1:
        if vector[5] == i[0]:
            total_area += vector[3] * vector[4]
            total_item_id += 1
            data_lis[index][1].append(vector[1])
            data_lis[index][2].append(vector[0])
    for mat_index, mat in enumerate(data_lis[index][1]):
        data_lis[index][1][mat_index] = mat / total_item_id
    data_lis[index].append(total_area)
    data_lis[index].append(total_item_id)

data = []
for v_index, vector in enumerate(data_lis):
    data.append([])
    for c_index, cri in enumerate(material_set):
        sta = 0
        for num in vector[1]:
            if num == cri:
                sta += 1
        data[v_index].append(sta)


def K_com(data, k_min, k_max, cluster_size):
    K_lis = range(k_min, k_max)
    S = []
    feasible_k_lis = []
    feasible_k = []
    for k in K_lis:
        C_kmeans = KMeansConstrained(n_clusters=k, size_min=cluster_size[0], size_max=cluster_size[1])
        C_kmeans.fit(data)
        result_label = C_kmeans.labels_
        S.append(metrics.silhouette_score(data, result_label, metric='euclidean'))
        data['cluster'] = C_kmeans.labels_
        print('聚类后，每个类别的个数\n', data.cluster.value_counts())

        total_area_lis = [0]*k
        total_item_num = [0]*k
        for label_index, label_num in enumerate(result_label):
```

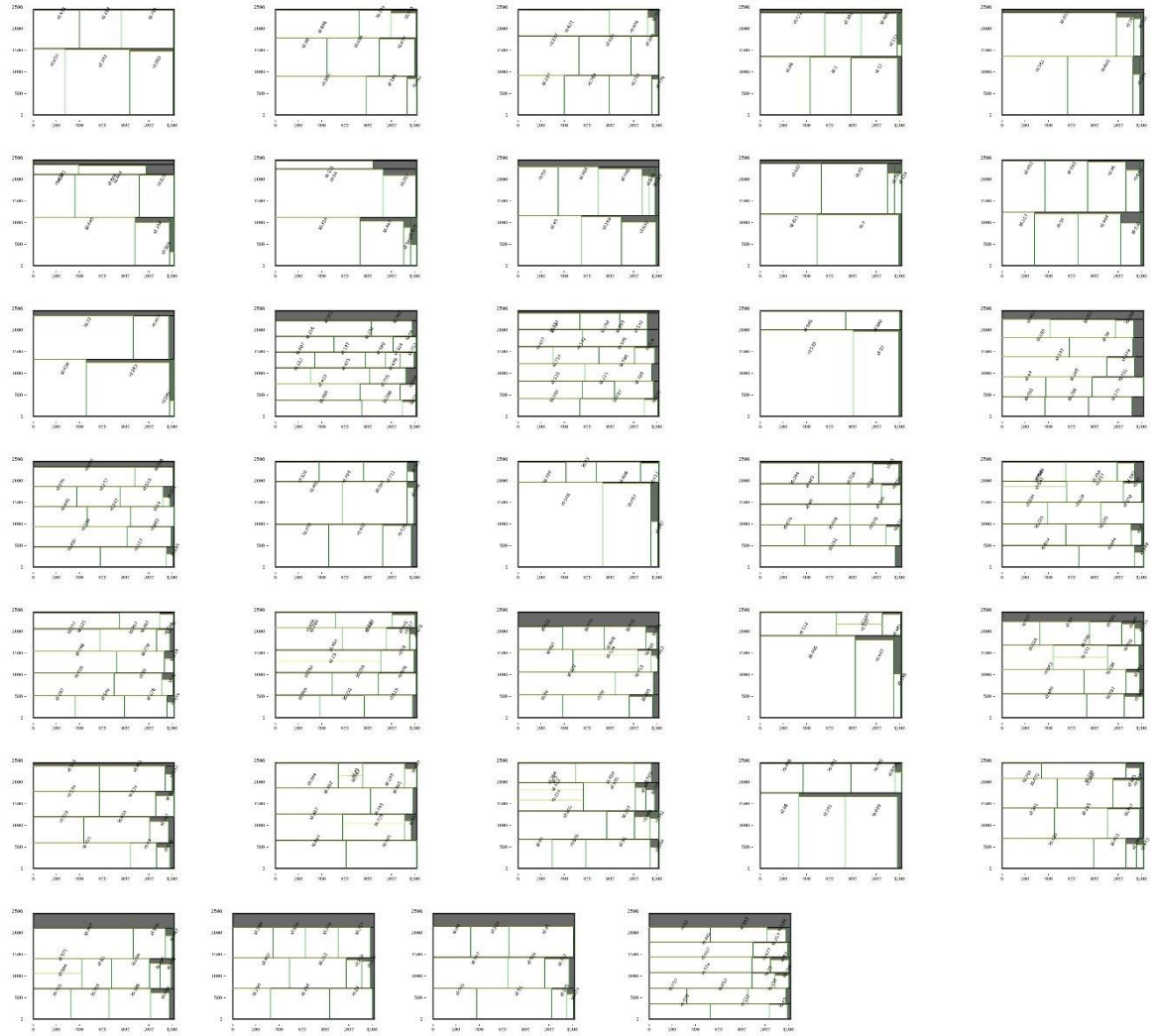
```
                total_area_lis[label_num-1] += data_lis[label_index][3]
                total_item_num[label_num-1] += data_lis[label_index][4]

        err_area_num = 0
        err_iteam_num = 0
        for cri in total_area_lis:
            if cri > 2.5e+8:
                err_area_num += 1
        for cri in total_item_num:
            if cri > 1000:
                err_iteam_num += 1

        print("有{0}个类超出了 250m2 的限制，有{1}个类超出了 1000 个的限制
".format(err_area_num, err_iteam_num))
        if err_area_num == 0 and err_iteam_num == 0:
            feasible_k_lis.append(k)
            feasible_k.append(result_label)

    print(feasible_k_lis)
    plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
    plt.rcParams['axes.unicode_minus'] = False
    plt.plot(K_lis, S, 'b*-')
    plt.xlabel('簇的个数')
    plt.ylabel('轮廓系数')
    plt.savefig(".\pic_B5.jpg")
    plt.show()

    return feasible_k_lis, feasible_k

data_pd = pd.DataFrame(data)
feasible_k_lis, feasible_k = K_com(data_pd, 210, 225, [2, 20])

for n_index, number in enumerate(feasible_k[0]):
    data_lis[n_index].append(number)
print(data_lis) # [order, material, id, total_area, total_item_id, cluster_index] id 和 material
为 list
for v_index, vector in enumerate(b1):
    for in_index, in_num in enumerate(data_lis):
        if vector[5] == in_num[0]:
            b1[v_index].append(in_num[5] + 1)

save_file = pd.DataFrame(b1)
save_file.columns=
["item_id","item_material","item_num","item_length","item_width","item_order","item_
area","item_area_10000","item_area_1000000","cluster_index"]
save_file.to_csv(r".\new_data_B5.csv", index=None, encoding='utf_8_sig')
```

附录 5

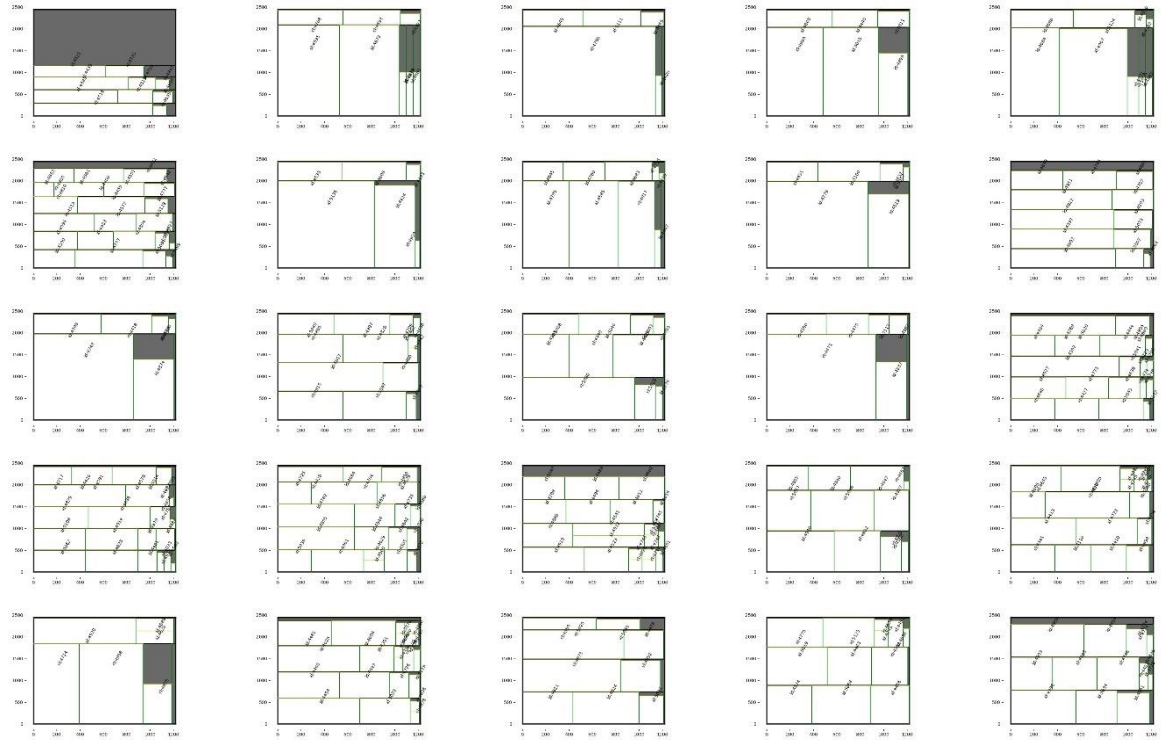
问题一，超参数为中位数输出排样方案效果图

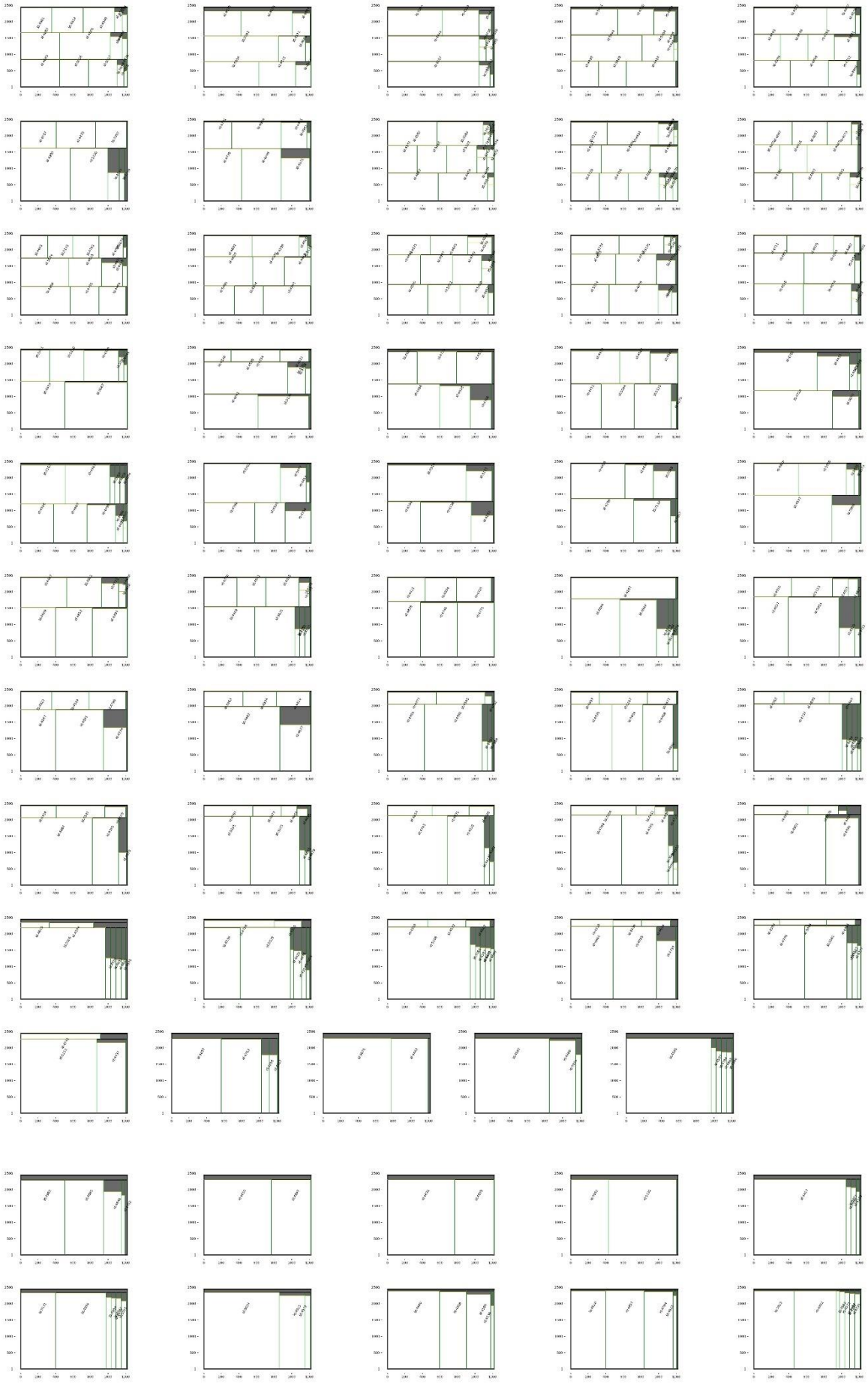
A1:

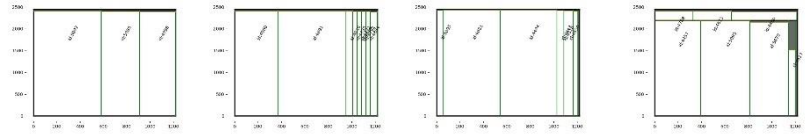




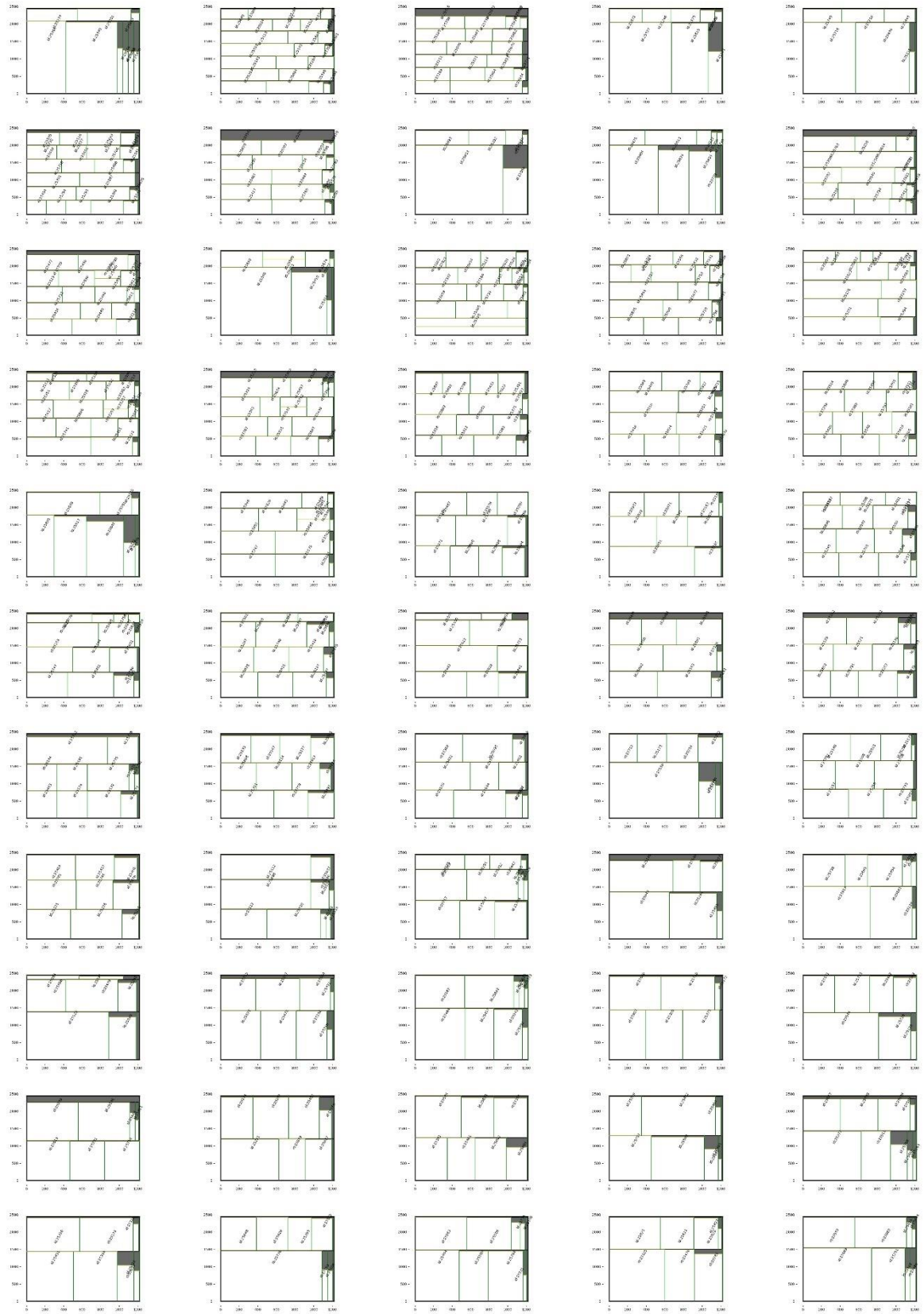
A2:

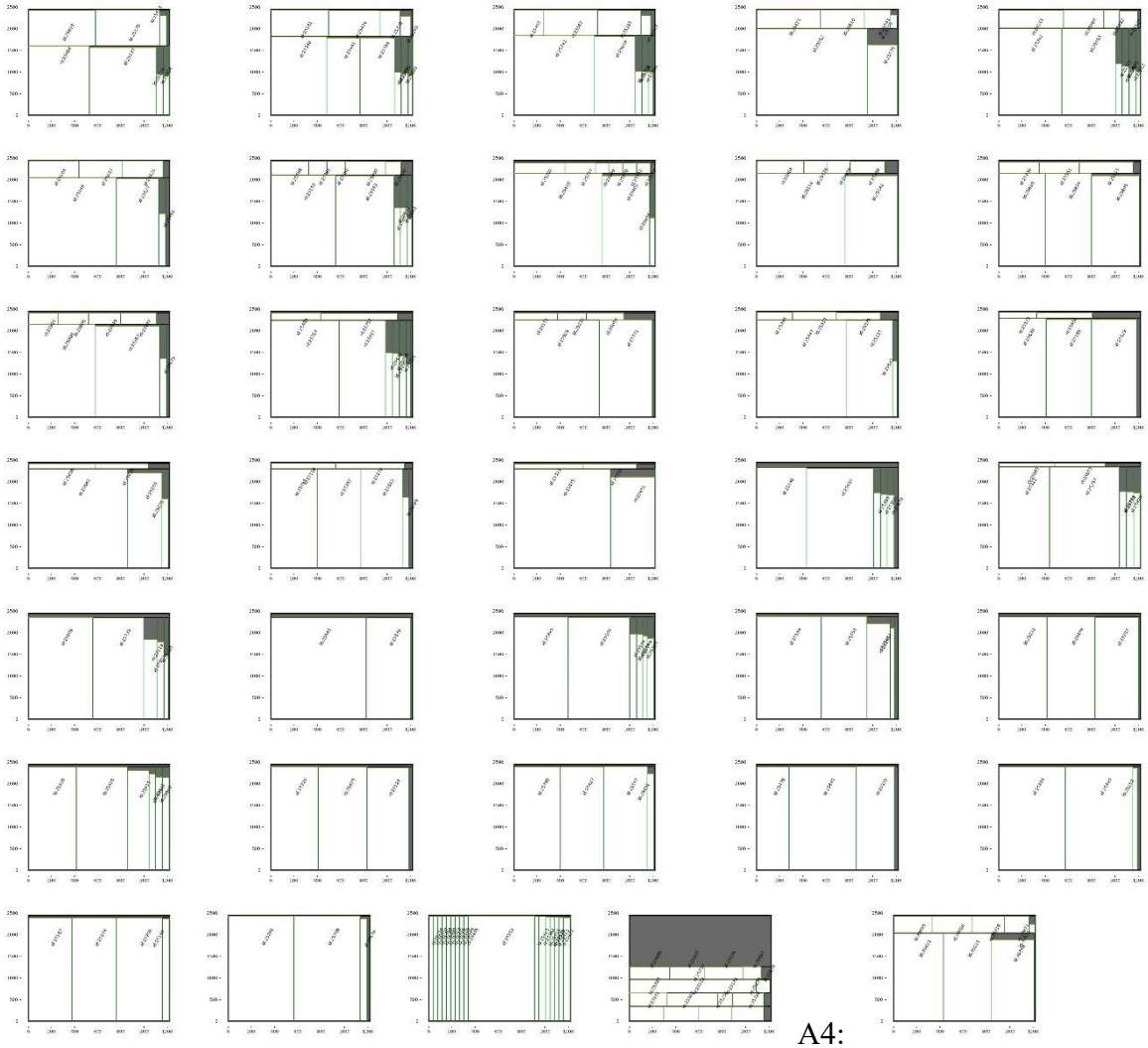






A3:





A4: