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建模与优化

摘　要

本文主要解决的是方形件生产过程中排样优化问题及订单组批问题。在排样优化问题中，我们从切割方式和排样方式两个角度分别建立了混合整数规划模型，运用粒子群算法和遗传算法对排样方式模型求解近似解。考虑到数据集的复杂程度，采用基于贪婪搜索算法的价值修正排列优化方案改进智能优化算法对模型进行求解，提升了优化算法的收敛速度和求解效率，得到较高板材利用率的排样方式结果。对求解算法结果进行对比分析，得出最终求解算法。

对于订单组批问题，根据题目所给约束条件，以所有批次总耗材最少为优化目标建立混合整数规划模型，对题目条件进行分析得到隐性评价指标，采用基于该评价指标的距离度量进行层次聚类，得到最优分批结果，再运用问题一中采用的改进优化算法对各批次订单进行排样优化。

关键词：PSO 算法 ；遗传算法 ； shuffle 全排列；贪婪搜索 ；层次聚类

目录

一、 问题重述

1.1 背景

《“十四五”智能制造发展规划》提出：“高度协同新一代信息技术与先进制造技术是先进智能制造的基础，其涉及到智能制造活动中的设计、分工、生产、销售等许多环节，形成了一种新型的生产方式，其具有自感知、自学习、自决策、自执行、自适应等功能”。智能制造的基本是数字化，实现智能制造的前提是企业应用数字化产品设计与工艺设计软件等工具，打造数字化的生产线、车间、工厂。因此，智能制造的主攻方向是制造业数字化转型，高速发展的智能制造会导致制造业制造范式、企业形态和产业模式产生颠覆性改变，大力促进制造业实现数字化转型。

基于物联网，大数据，人工智能等新兴手段，创立并完善智能化多级协同供应链体系，形成基于产能需求的供应链智能动态供给能力，完成内外创新资源、服务能力与生产能力的高度一体化，智能生产与服务运维信息的最大化协同，增强资源和服务的动态处理分析和柔性配置能力，完成供应链上下游企业更好的分工和合作，最终完成全产业链协同的最优能力配置。

在 2035 智能制造的新时代，方形件产品订单以要求生产数量大，产品品种多为特点，而“订单组批，批量生产”的生产模式不仅可以实现产品大批量生产同时也提高板材原片的利用率。**订单组批**问题与**排样优化**问题的解决是实现这种理想的生产模式的基础。订单组批问题是将不同订单组合起来以提高原材料利用率，提高生产效率为目的进行合理划分批次生产。排样优化问题是以降低板材原片消耗为目的对订单中方形件在板材原料上的切割分布进行排列。

综上所述，订单组批与排样优化可以让方形件生产过程减少对资源的浪费，加快生产效率，减少生产成本。对企业减少资源、成本、能源，提高生产效率有非常重要的意义。

1.2 问题

1.2.1 问题一

排样优化问题，建立混合整数规划模型，在满足生产订单需求和相关约束条件下，尽可能减少板材用量。

约束：

1) 在相同栈（stack）里的产品项（item）的宽度（或长度）应该相同；
2) 最终切割生成的产品项是完整的，非拼接而成。

1.2.2 问题二

订单组批问题。要求建立混合整数规划模型，对数据集 B 中全部的订单进行组批，然后对每个批次进行独立排样，在满足订单需求和相关约束条件下，使得板材原片的用量尽可能少。

在满足子问题 1 约束的基础上进一步要求：

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；
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。

二、 问题分析

2.1 问题一的分析

问题一本质上是一个二维三阶段一刀切的排料问题，其中原材料是规定尺寸且库存数足够多，而题目要求我们在满足题目中约束条件下，提高板材利用率。我们可以利用板材消耗最少这一目标函数建立混合整数规划模型进行求解。

值得注意的是订单需要切割的项目类型很多且几乎不同规格，这就导致了算法可能不容易收敛。试想一下当不考虑空间性，不考虑方形件的旋转且所有方形件在同一最低水平线下的情况下进行排列组合，以第一个订单 752 个方形件数据全排列为例，可能排列方式数目，经计算机计算如下图 1 所示。仅以此最简单的情况考虑，让计算机将所有排列方式遍历，计算成本就已经远远超出想象。

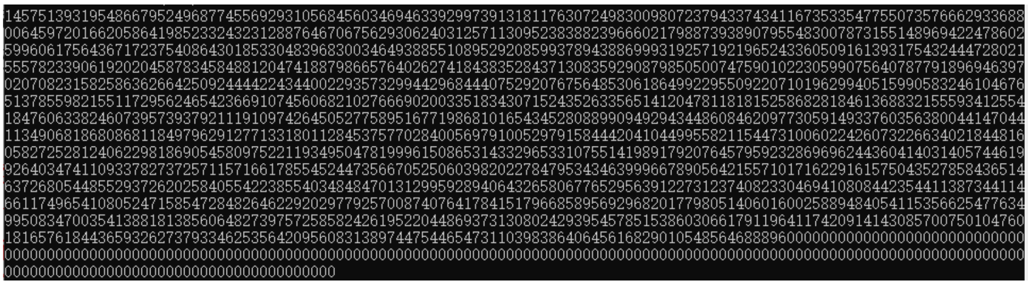

图 1 最简单情况下 752 个方形件全排列种类数目

所以我们在考虑算法模型时，会思考排列方案与优化方案以及算法的选择，但时间有限，我们仅考虑几种情况。初步逻辑框架图如下图 2 所示：

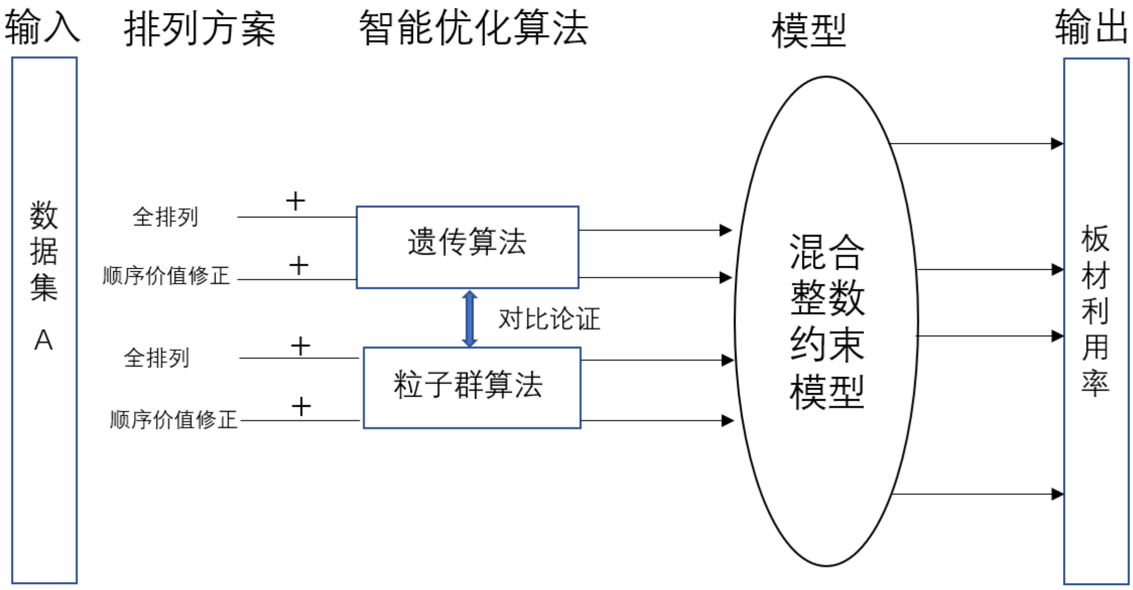

图 2 问题一求解逻辑框图

2.2 问题二的分析

问题二主要是一个订单组批寻优问题，我们需要寻找要求的组批原则。而题目已经给出了优化目标是使得板材原片的消耗最少。我们可以以这个优化目标和相关题目所给出的约束条件构建混合整数规划模型，采用启发式的智能优化算法进行求解。对优化目标进行分析，需要的板材数量最少，就要求我们将同种材料的订单最大化放进统一个批次中，且同一个批次中相同材料的原件越多，排样后的板材利用率就越高。

需要注意的是一个批次中的订单数并没有给出，这需要我们根据题目限制条件进行估计。批次分完就是和问题一相同的方法对每个批次进行排样优化的切割，这在问题一解决的基础上没什么难度。

三、 模型假设

因为方形件在切割过程会有切割损耗，切割完成也需要对切割边缘进行打磨等加工处理才能正常使用。于是我们对于题目做出如下假设：

1. 只考虑齐头切的切割方式（直线切割、切割方向垂直于板材一条边，并保证每次直线切割板材可分离成两块）；
2. 切割阶段数不超过 3，同一个阶段切割方向相同；
3. 排样方式为精确排样；
4. 假定板材仅有一种规格且数量充足；
5. 排样方案不用考虑锯缝宽度（即切割的缝隙宽度）影响；
6. 方形件切割加工时的损耗忽略不计。

四、 符号说明

符号	说明	单位
N	不同规格产品板的类型数目	个
M	剩余板的类型数目	个
d_i	产品板类型的需求量	个
x_{ij}	第 j类型的板上切割出第i个类型产品的数量	个
r	旋转方向	无
L,W	产品板的长度和宽度	M
$e(x,i)$	均质栈 $x\times w_i$ 中最多包含产品项 i 数量	无
$v(x,i)$	均质栈 $x\times w_i$ 中最大价值	无
e_i	均质栈 $x\times w_i$ 中包含有效产品项 i 数量	无
v_i	均质栈 $x\times w_i$ 中有效价值	无
$F(x,y)$	子栈 $x\times y$ 的价值	无
e_i	均质栈 $x\times w_i$ 中包含有效产品项 i 数量	无
a_{iq}	第 q 种排样方式包含的类型为 i 的产品板数量	个
E	订单数	个
C	产品板种数	个
$h_{zj}(1\le z\le n,1\le j\le e_z)$	第 z 个订单中第 j 个产品板	无
(l_{zj},w_{zj})	第 z 个订单中第 j 个产品板的长度	M
d_{zj}	第 z 个订单中第 j 个产品板的需求量	个
$\Omega_{f'}$	第 f'个批次$t_{f'}$所需要的板材数	个
b	订单	无
e	订单对应产品板个数	个
t	批次	无
$Area_{same}$	两个批次中相同的产品板面积总和	M^2
$Area$	两个批次产品板面积总和	M^2

Num_{same}	两个批次中相同的产品板数量总和	个
Num	两个批次中的产品板数目总和	个
$dist$	基于评价指标的距离度量	无

五、 模型的建立与求解

5.1 问题一排样优化模型的建立

5.1.1 排样优化特点及模型的引入

经过对问题 1 的分析，一块原始板材在经过三阶段切割过程中会有三种板子产生如图 3，在第一或第二阶段切割后，会产生产品板（订单中所需求规格的板），一部分是剩余板（还能再切割）。在经过第三阶段切割后会产生产品板和废料板（不能再进行切割）。

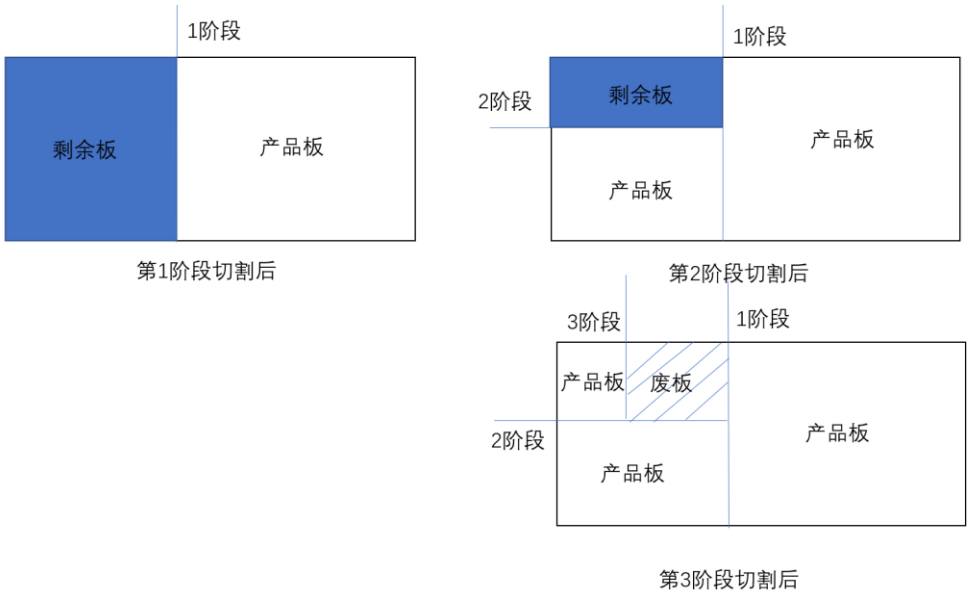

图 3 三阶段一刀切过程

因为常见的阶段最多为 3-4 个，因此以 3 阶段的切割方式为例（如图 4），第 1 阶段的横向切割生成模块称之为 stripe（条带），如 Stripe1 和 Strip2；第 2 阶段纵向切割生成模块称之为 stack(栈)，如 Stripl 继续被切割分成 Stack1、Stack2 和 Stack3；第三阶段横向切割生成模块称之为 item(产品项)，如 Stack1 继续被切割分成 Item1、Item2 和 Item3。

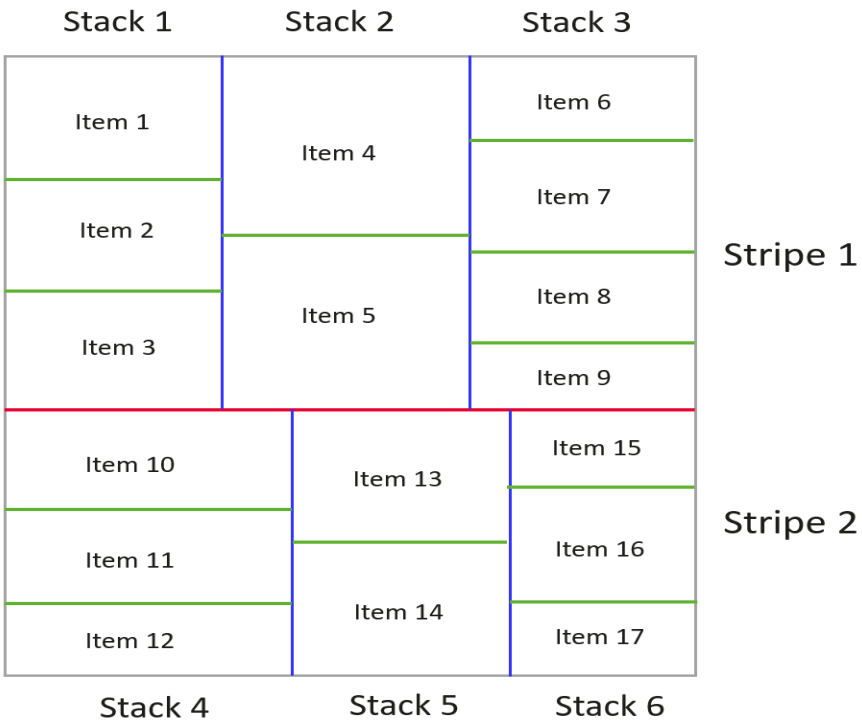

图 4 不同切割阶段的形式定义

5. 1. 2 模型的建立

本题我们分别从三段切割方式和排样方式两个角度出发，对该实际模型建立的两个数学模型如图 5 所示。

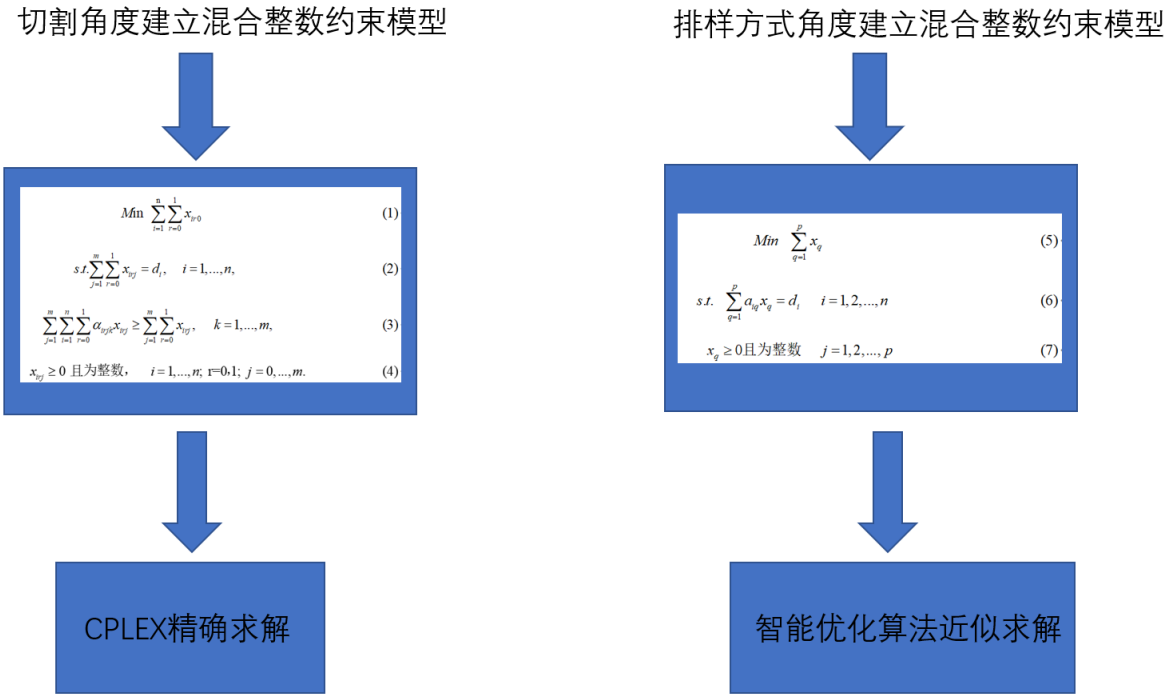

图 5 两个角度建立的模型

切割方式角度建立混合整数规划模型：从三段切割方式出发，考虑到本问题一刀切、三阶段的特点，我们的想法是将决策变量与每次一刀切的切割联系起来。限制条件考虑两方面，一方面满足需求规格，另一方面只能在现有的板材上进行切割。

我们定义不同规格产品板的类型数目为N，剩余板的类型数目定义为M。用j表示板的类型，$j=0$，1，2，…，m，其中$j=0$ 表示原始板，$j=1$，2，…，m对应剩余板的类型。用d_i表示产品板类型的需求量，$i=1$，2，…，n。定义决策变量x_{ij}表示在第 j类型的板上切割出第i个类型产品的数量。决策变量x_{irj}表示在第 j类型的板上切割出r方向的第i个类型产品板的数量，$r=0$表示产品板未旋转，$r=1$表示产品板旋转。如果当第k个板类型是从第j个板类型上切割下来的第i个产品板，则参数α_{irjk}为1，否则为0。

从三段切割角度建立了以耗材最少为目标函数建立混合整数规划模型如下所示：

$$Min \sum_{i=1}^{n}\sum_{r=0}^{1} x_{ir0} \tag{1}$$

$$s.t.\sum_{j=1}^{m}\sum_{r=0}^{1} x_{irj} = d_i, \quad i=1,...,n, \tag{2}$$

$$\sum_{j=1}^{m}\sum_{i=1}^{n}\sum_{r=0}^{1} \alpha_{irjk} x_{irj} \geq \sum_{j=1}^{m}\sum_{r=0}^{1} x_{irj}, \quad k=1,...,m, \tag{3}$$

$$x_{irj} \geq 0 \text{ 且为整数}, \quad i=1,...,n;\ \mathrm{r}=0,1;\ j=0,...,m. \tag{4}$$

耗材最少即所需原始板最少，式（1）表示以切割的原始板数目最少的目标函数，式（2）（3）（4）是约束条件。其中式（2）表示不允许生产出来的板子超过要求生产的板子数，式（3）表示只能切割现有的剩余板。式（4）约束了板子数目只能为非负整数。

排样方式角度建立混合整数规划模型：从排样方式角度出发，我们考虑设待切割的板材尺寸为$L\times W$，库存数量足够多，待排样的产品板有n种，相同尺寸的产品板即使订单 ID 不同，但我们将其看做一种型号产品板，给同一个型号。对于某个产品板，长宽以及需求量分别为（l_i, w_i, d_i），其中i为待排样产品板型号，$i=1,...,n$。根据题目要求确定合理三阶段排样方案，使用数量最少的板材完成订单产品板切割，以原始板材消耗最少为目标建立如下混合整数规划模型：

$$Min \sum_{q=1}^{p} x_q \tag{5}$$

$$s.t. \sum_{q=1}^{p} a_{iq} x_q = d_i \quad i=1,2,...,n \tag{6}$$

$$x_q \geq 0 \text{且为整数} \quad j=1,2,...,p \tag{7}$$

其中原始板材切割时的排样方式的集合为$O=[\mathrm{o}_1, o_2, ..., o_p]$共有 p 种排样方式，第q种排样方式所包含的类型为i的产品板数量为a_{iq}，整数决策变量x_q为第q种排样方式用在原始板上的次数。式（5）为目标优化函数，为让原始板耗材最少。式（6）表示不允许生产出来的板子超过要求生产的板子数，式（7）约束了决策变量只能为非负整数。

第一种角度的建模方法变量比较多，且第一种建模方式显然更适合使用商业软件进行求解。第二种角度的建模方式思路较为简单。我们可以单独考虑排样算法进行排

样方案的选择，从而再利用智能优化算法对模型进行求解。简单地说就是用排样算法优化智能算法来求解模型。详细描述见下一节。

5.2 问题一排样优化模型的求解

5.2.1 模型的特点以及算法引入

从排样方式角度出发，我们知道有 p 种排样方式，但是我们题目所给的订单数据量巨大。如果采用传统枚举算法寻找所有全排列排样方式的话，我们在问题一的分析中就提到仅考虑最简单的排列组合方式所需要的计计算复杂度就过高。当扩大可行解规模后，可行解数量呈指数级增长，采用精确算法求解该问题会出现维度灾难。综上所述我们采用全排列的思想，然后在编程中采用在约束条件逆向随机 *shuffle* 全排列堆叠的方式去完成一刀切的思想，然后利用智能优化算法来求解该问题，实现逆向随机 *shuffle* 全排列。

5.2.2 算法设计

总体算法设计：

1）数据预处理，导入数据集并且统一单位，将所有产品板中的长宽中较大的作为新的长度，较小的作为新的宽度；

2）将所有产品板（item）组成栈（stack）；

3）将所有栈（stack）组成条带（stripe）；

4）将条带（stripe）组成可行的排样方式；

5）智能优化算法根据优化条件对目标函数进行优化，主要采取采用逆向随机 *shuffle* 全排列堆叠的方式；

6）智能算法设计主要选取遗传和粒子群算法；

7）根据板材排样方式，计算排版方案数据，返回每一个产品板的排版数据。

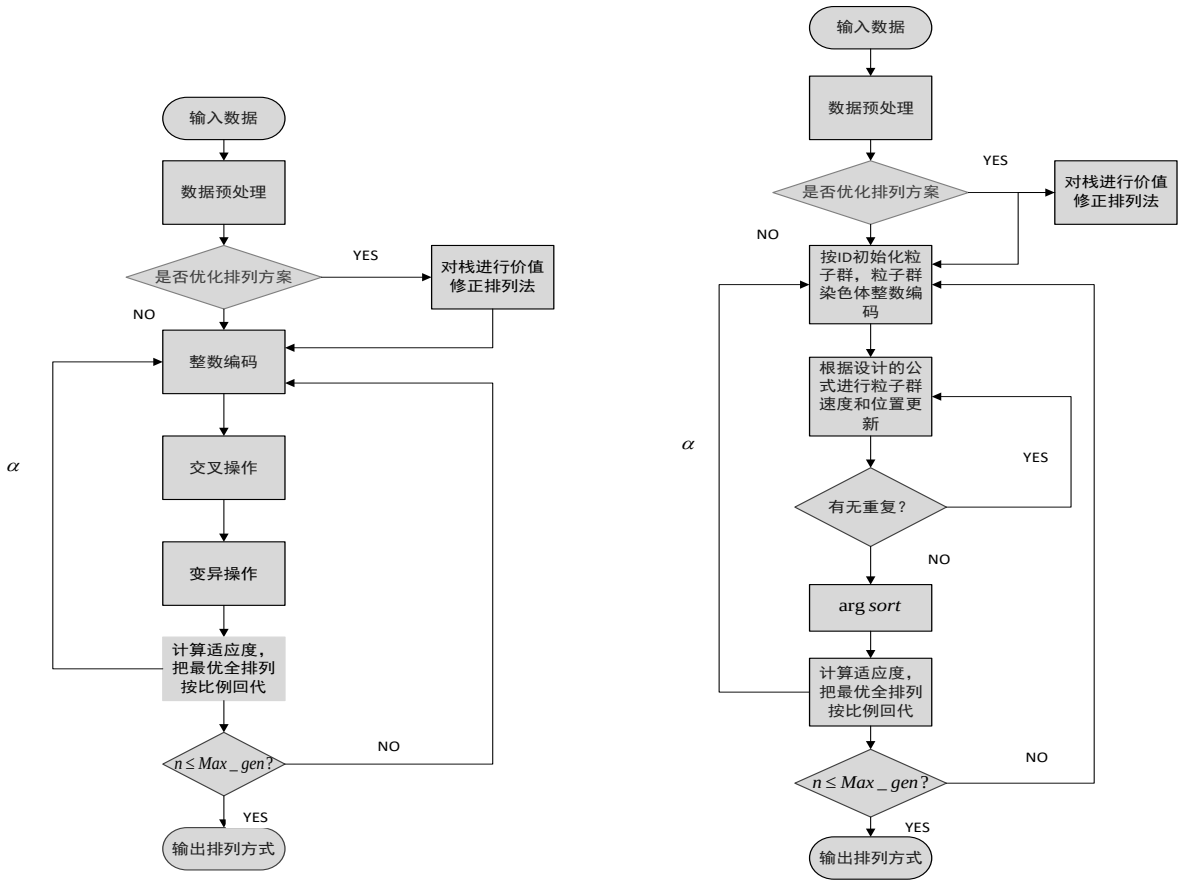

图 6 总体算法逻辑框图

逆向随机 *shuffle* 全排列堆叠：

为了实现一刀切最小剩料化，我们将随机全排列和物料切割的结合作逆向推导。全排列指从 n 个数中选取 m（m<=n）个数按照一定的顺序进行排成一个列，叫作从 n 个元素中取 m 个元素的一个排列。由排列的定义，显然不同的顺序是一个不同的排列。从 n 个元素中取 m 个元素的所有排列的个数，称为排列数。从 n 个元素取出 n 个元素的一个排列，称为一个全排列。排列数公式为：

$$A_n^m = n(n-1)(n-2)\cdots(n-m+1) = \frac{n!}{(n-m)!} \tag{8}$$

随机全排列算法如下图 7 所示：

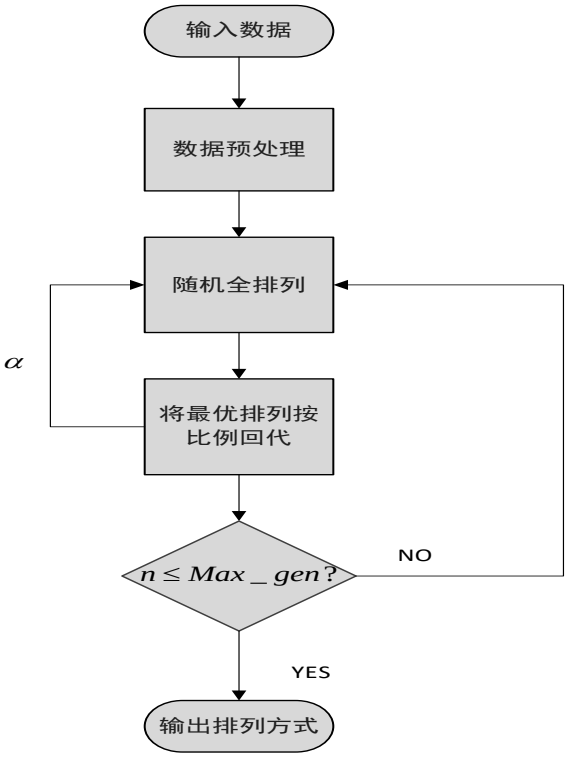

图 7 随机全排列算法图

遗传算法设计：遗传算法是借鉴生物的自然选择和遗传进化机制而开发出来的一种自适应全局优化概率搜索算法，它是模仿自然界生物进化机制发展起来的随机全局搜索和优化方法，借鉴了达尔文的进化论和孟德尔的遗传学说。其本质是一种高效、并行、全局搜索的方法，能在搜索过程中自动获取和积累有关搜索空间的知识，并自适应地控制搜索过程以求得最佳解。

原始算法逻辑：

Step 1. 初始化规模为 N 的种群并进行二进制或者整数编码。

Step 2. 对种群解码并评价每个种群的适应度。

Step 3. 对种群进行选择，保留前 X 个。

Step 4. 对种群按照概率 p_1 进行交叉操作，生成新个体加入种群。

Step 5. 对种群按照概率 p_2 进行变异操作，生成新个体加入。

Step 6. 返回 Step 2.

搜索过程以求得最佳解。

改进后实现 *shuffle* 全排列算法逻辑：

Step 1. 对输入产品的数据 ID 进行染色体整数编码，将可供组合的条带进行编号，然后染色体对应这些编号的一个完整的排列。

Step 2. 初始化规模为 N 的种群。

Step 3. 对种群按照概率 p_1 进行交叉操作，生成新个体加入种群。交叉操作在这里为交换两个染色体的部分排列顺序，比如两个染色体(1，2，3，4，5，6，7，8)和(8，7，6，5，4，3，2，1)进行交叉，可以按照概率随机选择几个位置，交换他们的排列，比如选择前三位进行交叉，则交叉后变为(6，7，8，4，5，1，2，3)和(1，2，3，5，4，8，7，6)。

Step 4. 对种群按照概率 p_2 进行变异操作，生成新个体加入。变异操作即以较低

概率交换一个染色体的部分位置，比如(1，2，3，4，5，6，7，8)变异操作可随机交换 3，5 两个位置，变为(1，2，5，4，3，6，7，8)最后输出结果即为较优的组合方式，根据这种组合方式组合相应的条带，确定板材的排样。

Step 5. 按顺序组合条带，不断将之后的条带拼接到之前的组合体中，直到长度超过样板长度为止，更换新样板。记录使用的样板数量和组合方式。评估个体适应度：以使用的板材数量作为该个体的适应度，使用板材越少则适应度越高，该个体越容易被保留。

Step 6. 对种群进行选择，保留前 X 个最优组合方式进行回代。

Step 7. 返回 Step 3.

算法流程图如下图 8 所示：

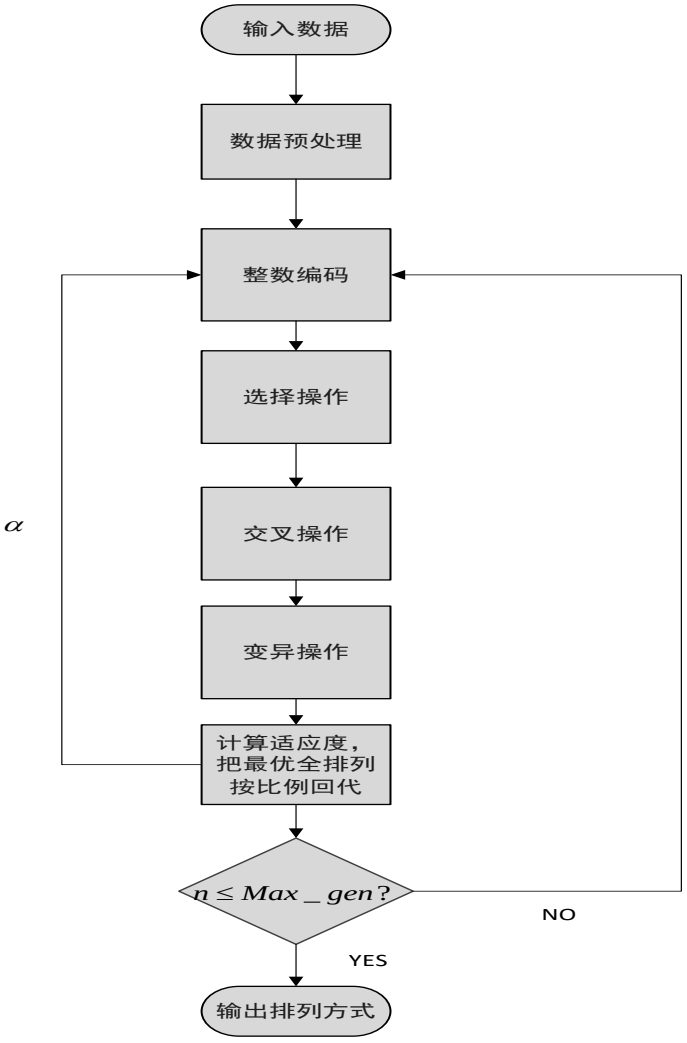

图 8 遗传算法逻辑框图

粒子群算法设计：

在粒子群（PSO）算法中，每个优化问题的潜在解都是搜索空间中的一只鸟，称之为“粒子”。所有的粒子都有一个由被优化的函数决定的适应值(Fitness Value)，每个粒子还有一个速度决定它们飞翔的方向和每一步的位移。然后粒子们就追随当前的最优粒子在解空间中搜索。PSO 算法需要初始化一群随机粒子(随机解)，然后通过迭代找到最优解，在每一次迭代中，粒子通过跟踪两个“极值”来更新自己。[1]

原始算法逻辑：

Step 1. 初始化规模为 N 的粒子群，每个粒子的速度和位置随机。

Step 2. 评价每个粒子群的适应值。

Step 3. 若某个粒子当前的适应值 *present* 比之前记录的该粒子最优解 *pbest* 更好，则更新 *pbest* 。

Step 4. 若某个粒子当前的适应值 *present* 比之前记录的全局最优解 *gbest* 更好，则更新 *gbest* 。

Step 5. 若 *gbest* 符合要求，则结束。若 *gbest* 不符合要求，粒子根据如下的公式来更新自己的速度和新的位置

$$\begin{aligned} v(k+1) &= wv(k) + c_1 r_1 (pbest(k) - present(k)) + c_2 r_2 (gbest(k) - present(k)) \\ present(k+1) &= present(k) + v(k+1) \end{aligned} \tag{9}$$

其中，是 k 时刻的速度，是 k 时刻的位置，和是服从伯努利分布的 0 或 1 随机数，是惯性权重因子，和称为加速因子或学习因子。

shuffle **全排列算法改进后算法逻辑：**

Step 1. 按 ID 初始化规模为 *N* 的粒子群，每个粒子的速度和位置随机。

Step 2. 对输入产品的数据 ID 进行粒子群染色体整数编码，将可供组合的条带进行编号，然后染色体对应这些编号的一个完整排列。

Step 3. 粒子群根据如下的公式来更新自己的速度和新的位置。

$$\begin{aligned} v(k+1) &= wv(k) + c_1 * r_1 * randi([\min, \max]) + c_2 * r_1 * randi([\min, \max]) \\ present(k+1) &= present(k) + v(k+1) \end{aligned} \tag{10}$$

Step 4. 判断每个粒子群中新的位置有无重合。

Step 5. 将每个粒子群的位置进行 $\arg sort$ 操作：每个粒子位置返回按从小到大的组合成的栈的数目远远小于产品板的

Step 6. 按顺序组合条带，不断将之后的条带拼接到之前的组合体中，直到长度超过样板长度为止，更换新样板。记录使用的样板数量和组合方式。评估个体适应度：以使用的板材数量作为该个体的适应度，使用板材越少则适应度越高，该个体越容易被保留。

Step 7. 对种群进行选择，保留前 *X* 个最优组合方式进行回代。

Step 8. 返回 Step 3.

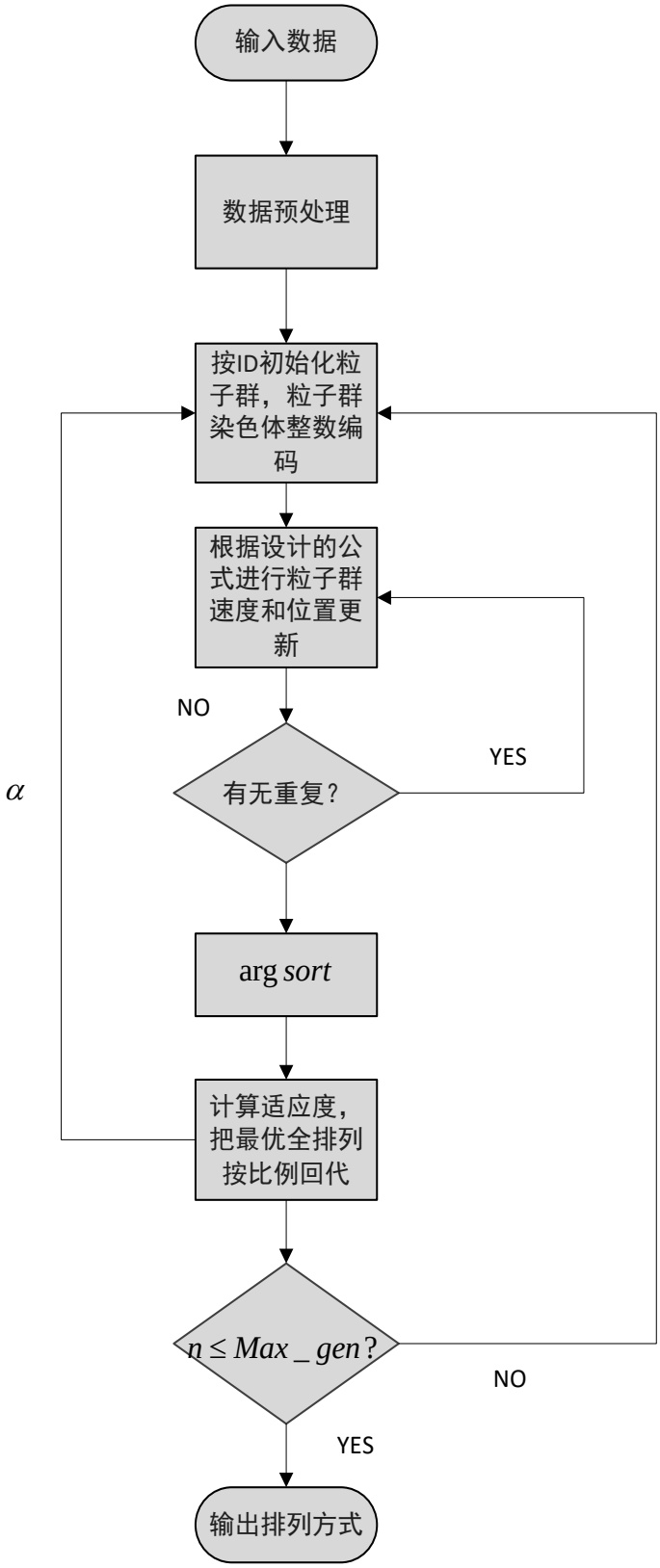

图 9 粒子群优化算法逻辑框图

如表我们尝试了仅仅通过两种智能算法对模型进行求解如表 1 所示，不仅花费时间庞大，而且板材利用率低。经过验证，确实和猜想的一样，仅通过智能优化算法进行求解易陷入局部最小值且计算复杂度较高。

表 1 仅使用智能优化算法对模型求解的效果

智能优化算法	模型求解时间	结果指标【利用率】	计算复杂度
遗传算法	4.725362s	55.69%	$O(kn^2)$
粒子群算法	6.129554s	54.96%	$O(n^3)$

5.2.3 智能优化算法的改进

但就像之前所说的，样本数据量过大，即使采用智能优化算法也不可避免在一定程度上遍历了所有数据从而不易达到最优值，容易陷入局部最小值，算法收敛慢。所以我们考虑增加排布方式算法，给智能优化算法一个策略，引导算法收敛。初步思想如下图 10 所示，我们采用价值修正思想对栈进行了优化排列作为算法收敛的引导策略。价值修正思想其实就是利用生活中人工排样的经验演化而来，工人优先会选用大的产品板进行组合，因为这些尺寸远远大于其他产品板的板子与别的板子很难组合成好的排样结果。如果采用原始的排列方案对所有产品板进行全排列随机优化，很容易陷入局部最小，无法得到好的排样方式。

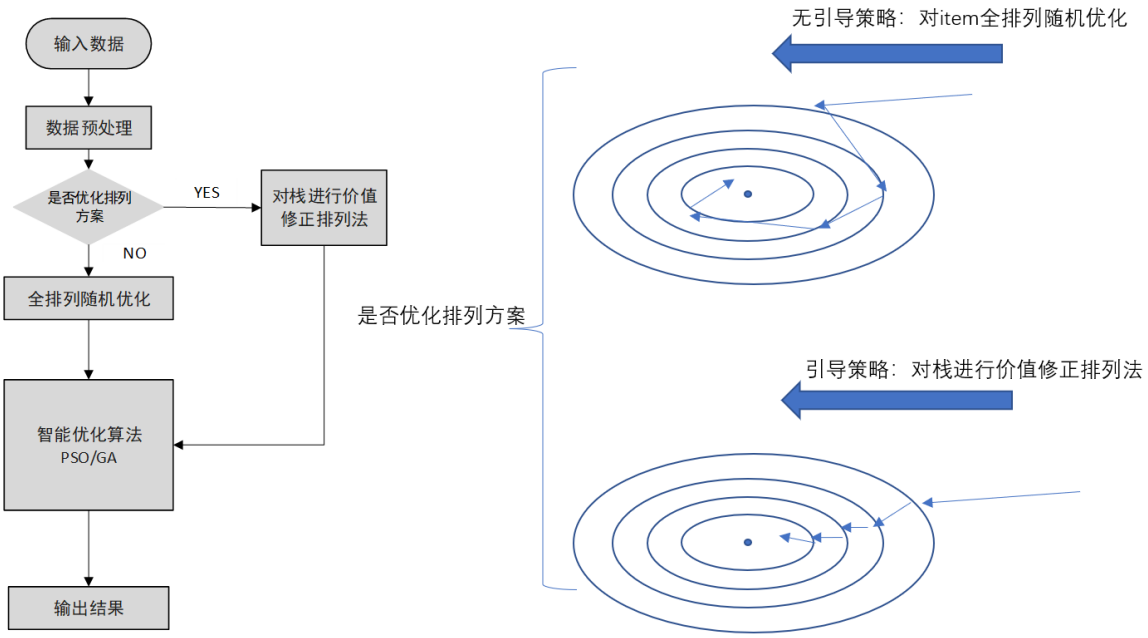

图 10 智能优化算法初步改进思想

具体价值修正排列算法如图 12，13 所示，我们考虑到产品板类型数目很大，所以我们还考虑了优先将产品板（item）组合成栈（stack），将栈按面积价值顺序排序再组合成条带（stripe）。这样不仅满足了题目要求三段一刀切的约束，而且由产品板组合成的栈的数目远远小于产品板的个数，排列的方式数目也呈倍数下降，使得智能优化算法不易陷入局部最小。

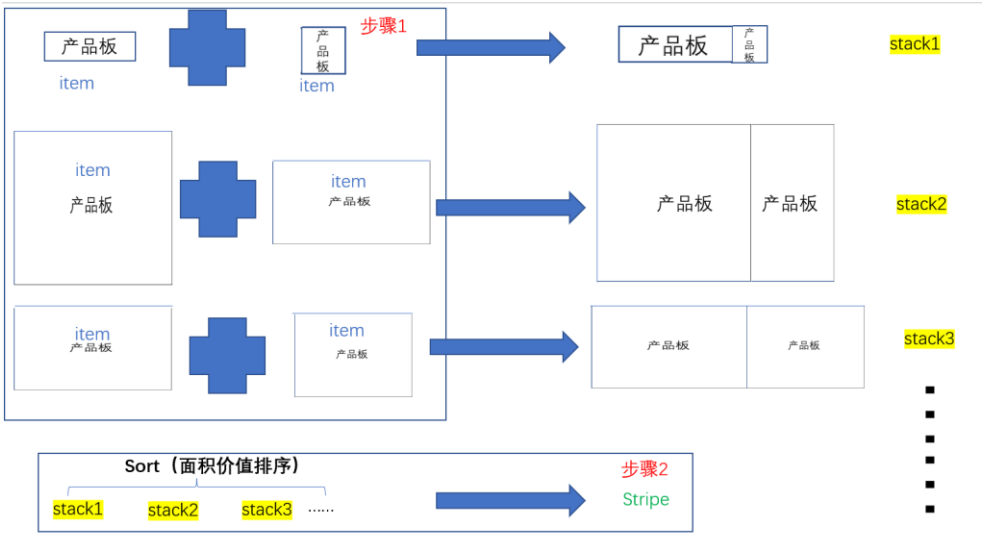

图 11 价值修正排列算法步骤 1 与 2

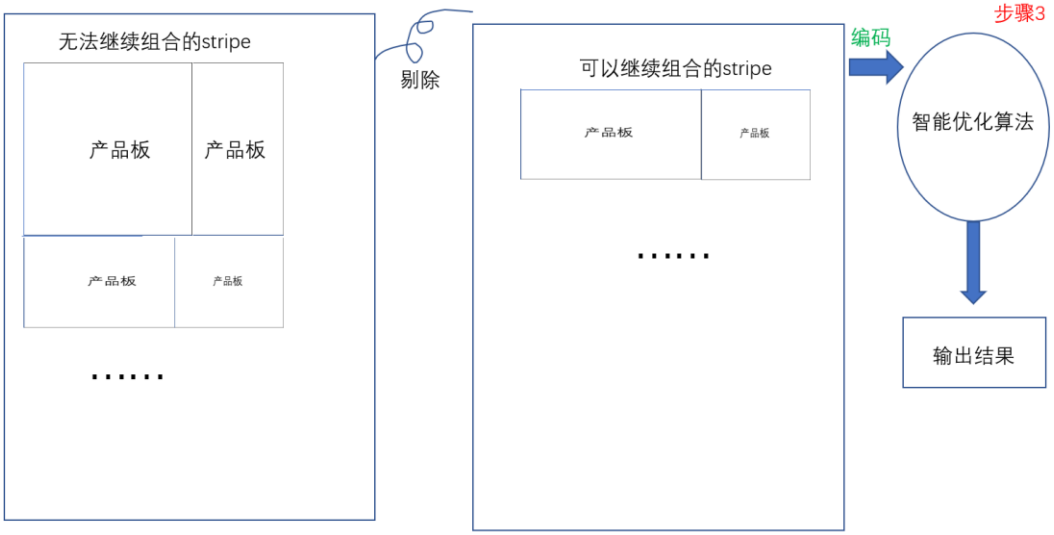

图 12 价值修正排列算法步骤 3

价值修正排列算法[5]改进智能优化算法的具体实现步骤如下：

Step1：将所有产品板（item）组成栈(stack)；

1）计算每一类产品项 item（产品项）的数量和其长度、宽度(所有数据均。更大的边长为长，更小的边长为宽）。

2）我们优先将同类产品拼接组成栈，也就是题目中提到的同质排样方式。

3）然后再考虑如果有剩余的均值排样，也就是某个产品板的长或者宽等于其他产品，但是另一个并不相等，即不同规格的产品项组成栈。这里我们也考虑到了旋转的方式，我们对产品板增加负号表示旋转，然后进行重新运行。为了加快搜索速度和全局搜索能力，我们考虑了一部分贪婪算法的思想如图 9 所示，先找到剩余产品项中面积最大的产品项，让他单独成栈，放置在条带的最左边，固定面积最大的宽度为整个条带的宽度，让面积最大的产品项再与其他产品项组合成栈。然后条带的右边剩下的部分，我们继续寻找能够放入的面积最大的产品项.如果没有产品项能够放入，则条带输出，如果能放入则进行放入.放入新的产品项之后，看上面能否放入等

长的产品项，这里是均质排布，搜索有没有长度相等，同时能够放入上面的产品项，如果有，那么这两个产品项组成一个栈，如果没有，那么右边的产品项也是单独成栈.然后再向右搜索，直到右边的面积不够放下任何一个产品项，输出条带。

(4) 这里考虑的是，如果能组成栈的情况，尽量组成栈，直至再添加新的产品项

时，该栈的宽度超过原始板材的宽度.那么即使新的产品项长或宽与这个栈是一样的，那么也新开启一个栈来进行排布.也就是说，最极限的情况下，整个原始板材只有一个条带。

(5) 这一步的最后，我们得到了一系列的栈。

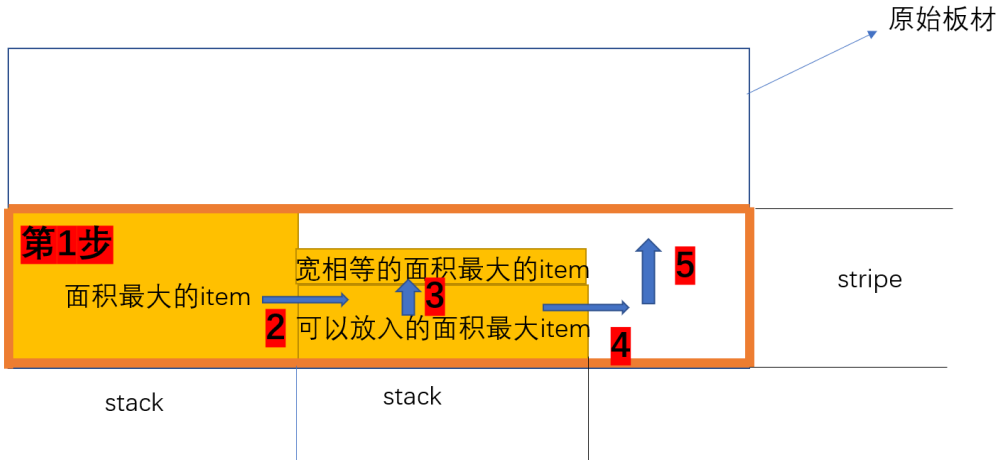

图 13 贪婪搜索算法

Step2：对第一步的栈进行编号，未优化前我们是对产品项进行全排列，此时是对栈

进行全排列。根据价值修正算法原理，因为长度相差较大的栈组成条带会导致浪费，所以按照栈长度由大到小的顺序依次组成条带组成条带时，按照栈长度由大到小的顺序依次组合，直到达到长度限制下的条带长度最大值，也就是栈长度之和不能超过板材长度.其中栈形成条带的约束条件是栈长度之和不能超过板材长度。

Step3：该步骤采用遗传算法或者粒子群算法，增加了一步数据预处理的操作，先剔

除无法与其他任何条带组合的条带，他们各自占据一个原始板材.然后再对剩余的部分进行组合。现在剩下的条带进行遗传算法的操作。很明显，现在只需要对条带进行一个全排列，数据量要比对产品项进行全排列少的多，也更易解出最优解。而智能优化算法的编码解码就是把产品项的编号换成做好的条带的编号。

Step1 将所有产品板（item）组成栈(stack)的正价值修的基本公式化：

(1)长度规范化

排样方式生成过程中只考虑一些特殊的离散切割位置不会影响到解的质量。3H 排样方式中所考虑的离散切割位置分布在长度方向上，因此又称为规范长度。3H 排样方式的规范长度应该大于为产品项长度的整数倍，

$$L=\left\{x_{ik}\left|x_{ik}=kl_i,0\le k\le[ri,\lfloor L/l_i\rfloor],k\in N,i=1,...,m\right.\right\} \tag{11}$$

L 为规范长度，其中 M 为元素的数量，l_i 为 i 型产品项组成的均质栈长度，生成 3H 排样方式时，只考虑规范长度的条带和栈，可以缩短许算时间而不影的排样方式的价值，后面涉及的皮带和栈都是指规范长度的条带和栈。

(2)栈生成

先用贪婪算法将剩余产品项中面积最大单独成栈放在左下角，对于 i 型产品项组成的均质栈 $x\times w_i, x\in L, i\in I_x, I_x\in\left\{i\left|l_i\le x\right.\right\}$，其所包含的产品项数量最多为 $e(x,i)=\min(\lfloor x/l_i\rfloor, r_i)$ 价值最大为 $v(x,i)=e(x,i)v_i$，可能的产品项数量为 e_i，$e_i=1,\cdots,e(x,i)$，可能的栈价值为 e_iv_i。

(3)条带生成

条带 $x\times W$，$x\in L$ 由水平条带沿垂直方向组合而成，其目标是最大化条带中所包含的条带价值之和。约束条件包括宽度约束和产品项需求约束:包含在条带中的条带宽度之和不能超过板材宽度，条带中包含的产品项数量不超过当前的剩余需求量。

条带中包含的栈可以由子栈三阶段一刀切的堆叠而得。如图 13 所示，子栈超过板材宽度，栈中包含的产品项数量不超过当前的剩余需求量。

条带中包含的栈可以由子栈三阶段一刀切的堆叠而得。如图 13 所示，子栈 $x\times y$ 可由子栈 $x\times(y-w_i)$ 和均质栈 $x\times w_i$;向上拼接而得。设子栈的价值和所包含的产品项数量分别为 $F(x,y)$，$n(x,0,i)=0$，$x\in L$，i=1，…，m.采用三阶段一刀切的堆叠方式可从 y=0 开始递推求子栈的价值 $F(x,y)$，确定子栈中的栈组合。

初始时，$F(x,0)=0$，$n(x,0,i)=0$，$x\in L$，i=1，…，m 由于有产品项需求的约束，子栈中包含的产品项数量不能超过其剩余需求量。子栈 $x\times(y-w_i)$)中已含有的型产品项数量为 $n(x,y-w_i,i)$ 因此栈 $x\times w_i$ 中 i 型产品项的有效数量只有 $e_i=\min\left\{\lfloor x/l_i\rfloor, r_i-n(x,y-w_i,i)\right\}$，$i\in I_y$，其中 $I_y=\{i\left|i\in I_x, w_i\le y\right.\}$ 为可排入的产品项集合。

$F(x,y)=\max\{F(x,y-1),\max_{i\in iy}[F(x,y-w_i)+e_iv_i]\}$，　$0<y\le W$

子栈 $x\times y$ 的初始解和子栈 $x\times(y-1)$ 的解相同。如果拼接得到的子栈价值比初始价值高，则采用拼接结果。采用三阶段一刀切的递推得到子栈的价值后，可按如下规则确定子条带中包含的各类型产品项数量:

若 $F(y)=F(y-1)$，则 $n(x,y,i)=n(x,y-1,i)$,，i=1，…，m

否则，若 $F(y)=F(y-w_k)+e_kv_k$，则:

$n(x,y,k)=n(x,y-w_k,k)+e_k$，$n(x,y,i)=n(x,y-w_k,i)$，$i\in I$，$i\ne k$ 最终得到的 $F(x,W)$ 即为栈 $x\times w_i$ 的价值，所包含的 i 型产品项数量为 $n(x,W,i)$，i=1，…，m

5.2.4 算法优化效果验证

我们这里以对数据 A1 进行排样优化求解为例，运用排列优化算法改进遗传算法和粒子群算法得到运行结果如下表所示。可见价值修正排列算法改进遗传算法比起价值修正排列算法改进粒子群算法更易收敛。

同时通过与未经过排列优化算法改进的智能算法求解效果对比发现，排列优化算

法确实在算法收敛上起到一个引导策略。

表 2 优化后智能优化算法对模型求解的效果

智能优化算法	模型求解时间	结果指标【利用率】	计算复杂度
遗传算法	3.725362s	87.02%	$O\left(kn^2\right)$
粒子群算法	7.129554s	87.02%	$O\left(n^3\right)$

下图 14-17 分别展示了按最终利用切割板数为损失的迭代图与样式排版图(部分)，如下图所示粒子群算法在收敛速度上比遗传算法弱，且我们可以清楚的发现粒子群算法容易陷入局部最优值。 综上我们在问题一与问题二的排样优化问题的求解上最终采用在贪婪搜索思想上的价值修正排列算法改进的遗传算法。

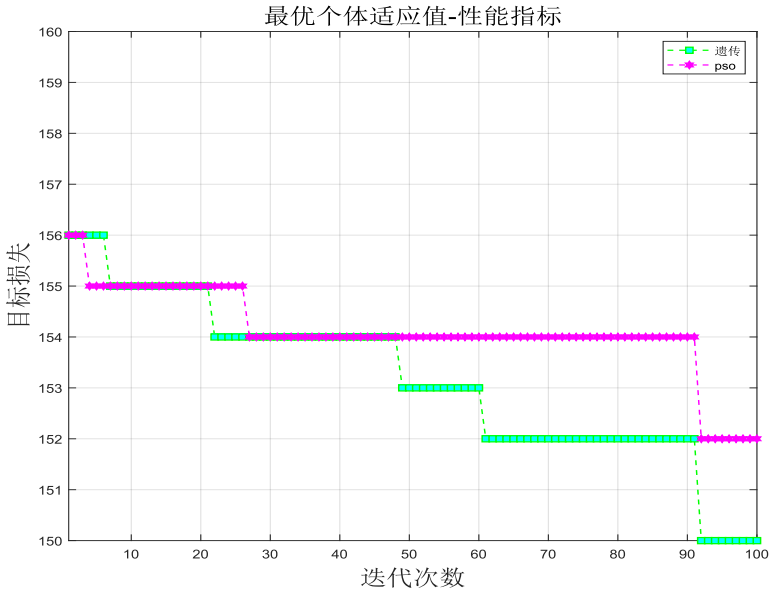

图 14 (优化前 loss)

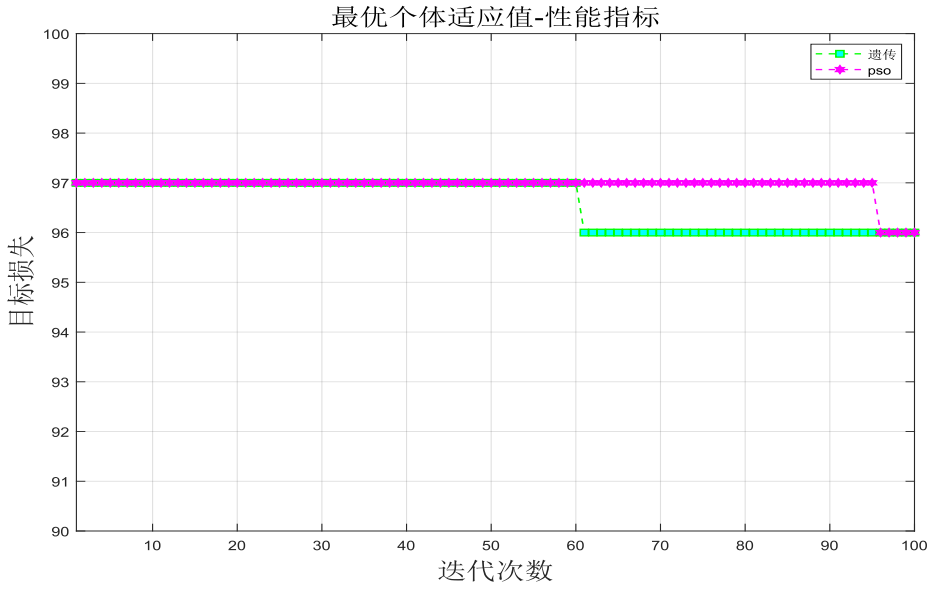

图 15 (优化后 loss)

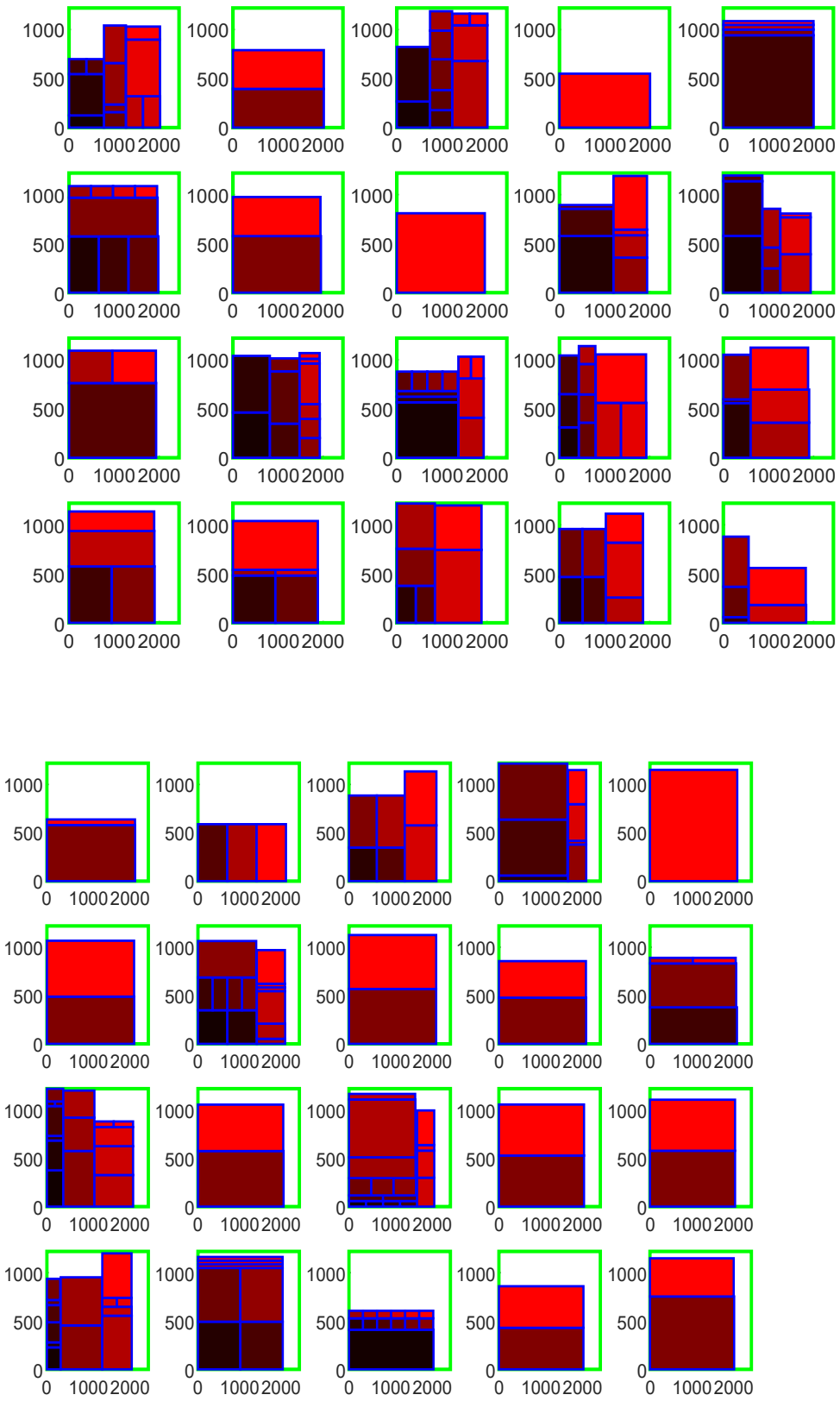

图 16 (优化前一刀切部分排版)

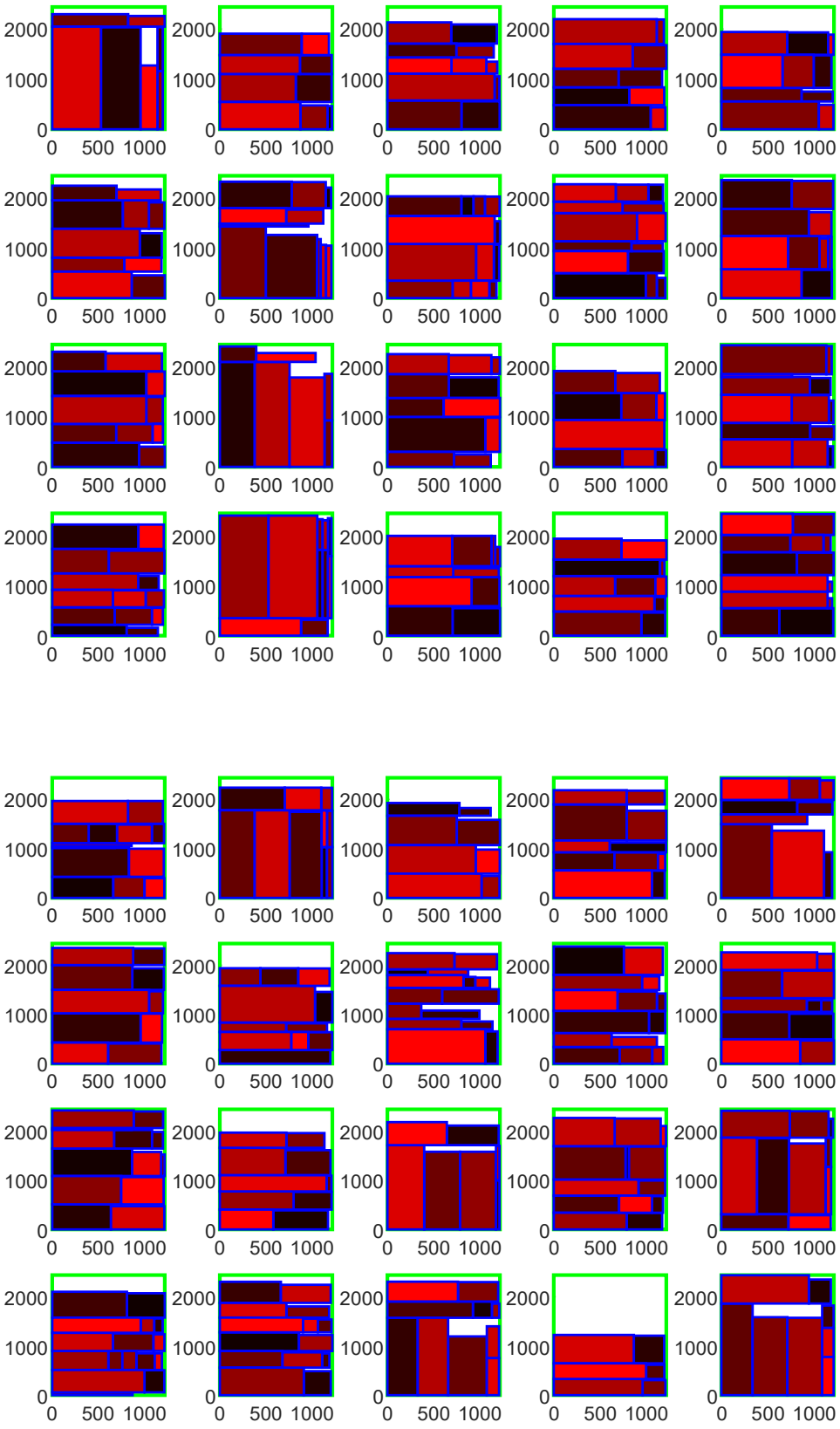

图 17 (优化后一刀切部分排版)

六、 问题二订单组批模型建立与求解

6.1 订单组批模型的建立

以板材消耗最少为目标函数建立混合整数规划模型，假设总共有 E 组订单 $(b_1,b_2,...,b_E)$，对应的订单有 $(e_1,e_2,...,e_n)$ 个产品板，产品板总共有 C 种材料。$h_{zj}(1\le z\le n,1\le j\le e_z)$ 表示在第 z 订单中第 j 个产品板，这个产品板的长度和需求量为 (l_{zj},w_{zj},d_{zj})。当第 h_{zj} 个产品板使用了第 c 种材料，$g_{zjc}=1$，否则 $g_{zjc}=0$。设所有订单分为 f 个批次 $(t_1,...,t_f)$，当第 z 个订单在第 f 个批次里，$g'_{zj}=1$ 否则 $g'_{zj}=0$。设 $\Omega_{f'}$ 为第 f' 个批次 $t_{f'}$ 所需要的板材数，则优化目标所有批次的原石板材为：

$$Min\sum_{f'=1}^{f}\Omega_{f'} \tag{11}$$

根据题目所给要求设置约束条件：

约束条件一：一个订单必须且只能在同一个批次中，因此 z 固定，所有 g'_{zj} 相加为 1，即：

$$s.t.\sum_{z=1}^{n}g'_{zj}=1 \tag{12}$$

约束条件二：每个批次所能生产的最多产品项个数是有上限的，所以当 j 固定，所有 g'_{zj} 乘以 e_z 需要小于题目所给上限 1000，即：

$$\sum_{z=1}^{n}g'_{zj}\,e_z\le 1000 \tag{13}$$

约束条件三：每个批次所能生产的产品项的面积是有上限的，每个订单中的产品板面积为 (l_{zj},w_{zj},d_{zj}) 三项相乘.对于一个批次来说，固定 f，把所有 g'_{zj} 乘每个订单的总面积，应该是小于题目所给产品板面积的上限 250 平方米，即

$$\sum_{z=1}^{n}g'_{zj}\sum_{j=1}^{e_n}l_{zj}w_{zj}d_{zj}\le 250 \tag{14}$$

根据上述优化目标和约束条件建立混合整数规划模型如下：、

$$Min\sum_{f'=1}^{f}\Omega_{f'} \tag{11}$$

$$s.t.\sum_{z=1}^{n}g'_{zj}=1 \tag{12}$$

$$\sum_{z=1}^{n}g'_{zj}\,e_z\le 1000 \tag{13}$$

$$\sum_{z=1}^{n}g'_{zj}\sum_{j=1}^{e_n}l_{zj}w_{zj}d_{zj}\le 250 \tag{14}$$

其中在计算所有批次耗材 $\sum_{f'=1}^{f}\Omega_{f'}$ 的过程中，每一个产品板都需要通过乘 g'_{zj} 来判

断是否属于这个批次，还要乘 g_{zjc} 判断是否属于同一材质，最后对于不同材质在进行求和。

6.2 订单组批模型的求解

首先我们对订单组批模型进行分析，订单分批原则就是考虑订单交货期、设备产能负荷、仓储容量、材料利用率、生产效率、生产工艺约束等因素进行分批。简而言之就是比较每个订单（或批次）之间的相关性，将相关性强的订单（或批次），我们将他们组合成在一个批次。

于是我们采用层次聚类的方法进行分批次，再对每个批次进行问题一的求解算法。

6.2.1 基于评价指标的距离度量方法

题目要求一块原始板材只能切割出一种材料的产品板，那么就可以得到一个隐含评价指标，只要让两个批次 t_{z1},t_{z2} 中同种材料的产品板数和面积都足够大，排样后的板材利用率也就越高。因此，我们在对两个批次进行距离度量时，可以将这个隐含评价指标作为距离度量的一种方式，得到基于评价指标的距离度量公式如下：

$$dist(t_{z1},t_{z2})=1/(\lambda_1\frac{Area_{same}(t_{z1},t_{z2})}{Area(t_{z1},t_{z2})}+\lambda_2\frac{Num_{same}(t_{z1},t_{z2})}{Num(t_{z1},t_{z2})}) \tag{15}$$

其中 $\lambda_1+\lambda_2=1$，$Area_{same}$ 是两个批次中相同的产品板面积总和，$Area$ 是两个批次产品板面积总和。Num_{same} 是两个批次中相同的产品板数量总和，Num 是两个批次中的产品板数目总和。

我们在组批过程中还要遵循批次合并后的面积和数量都不能超过题目要求的批次面积和数目的上限。于是我们给出完整的距离度量公式：

$$dist(t_{z1},t_{z2})=\begin{cases}1/(\lambda_1\frac{Area_{same}(t_{z1},t_{z2})}{Area(t_{z1},t_{z2})}+\lambda_2\frac{Num_{same}(t_{z1},t_{z2})}{Num(t_{z1},t_{z2})}) & Area(t_{z1},t_{z2})<250\,|\,Num(t_{z1},t_{z2})<1000\\ \infty & else\end{cases} \tag{16}$$

$$d_{\min}\left(t_i,t_j\right)=\min_{x\in t_i,y\in t_j}dist\left(x,y\right) \tag{17}$$

$$d_{\max}\left(t_i,t_j\right)=\max_{x\in t_i,y\in t_j}dist\left(x,y\right) \tag{18}$$

6.2.2 基于层次聚类法的分批算法

层次聚类算法就是将每一个订单样本看做初始的批次，在每一次的迭代中，计算出距离最近的两个批次组合起来，直到达到批次的上限。事实上，如图所示，两个批次就是两个订单样本的集合 t_{z1},t_{z2}，我们只需要根据上述的距离度量计算出每个批次的距离，遍历所有距离找到距离最小的两个批次组合成一个新批次，不断循环分批。

具体算法步骤如下：

Step1:将所有订单单独组成批次，计算距离 $dist$ 得到距离矩阵 D，找出距离最近的两个批次 t_{i*},t_{j*}；

Step2：合并t_{i*},t_{j*}：$t_{i**}=t_{i*}\bigcup t_{j*}$；

Step3：比较合并后的t_{i**}最短距离是否无限大，如果无限大，移除t_{j*}，批次分批完成。否则返回 Step1，直至所有批次分完。

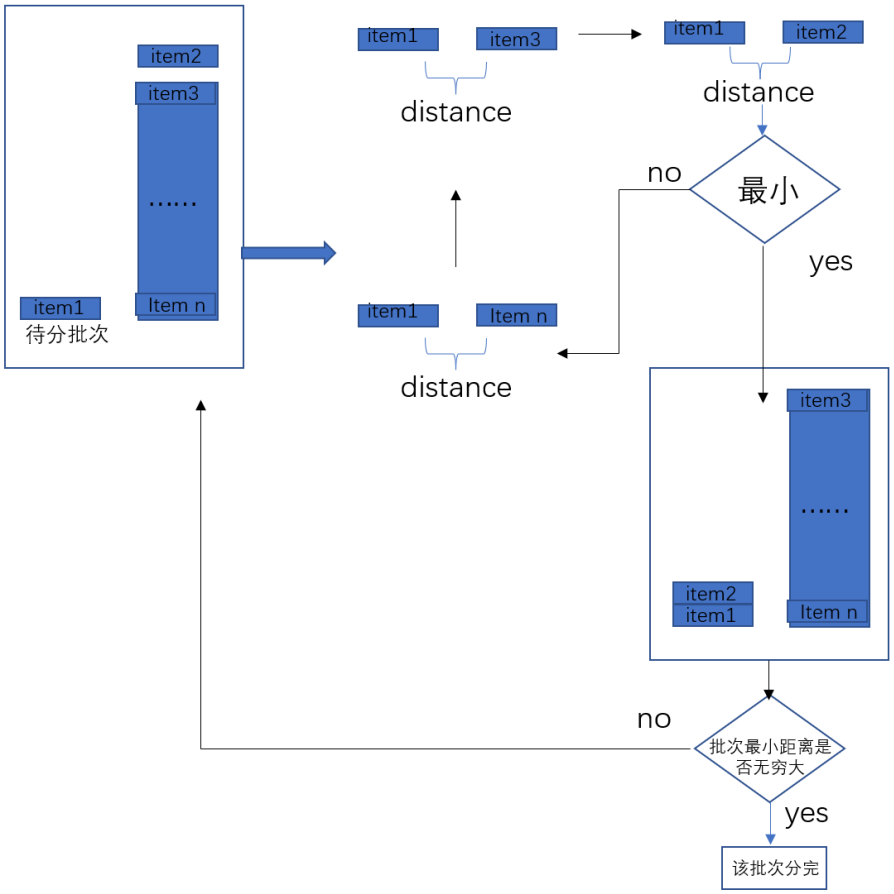

图 18 分批算法过程图

Algorithm ： 层次聚类

Input:

样本集$\mathcal{D}=\{x_1,x_2,...x_m\}$

距离度量系数d

聚类批次数k

1: for $j=1,2,...,m$ **do**

2: $t_j=\{x_j\}$

3: end for

3: for $i=1,2,...,m$ **do**

4: for $j=i+1,...,m$ **do**

5: $M(i,j)=d(t_i,t_j)$

6: $M(j,i)=M(i,j)$

7: end for

8: end for

9: 设置当前聚类批次个数： $q == m$

10: While $q < k$ **do**

11: 找出距离最近的两个聚类批次 t_{i*}, t_{j*}

12: 合并 t_{i*}, t_{j*}： $t_{i*} = t_{i*} \bigcup t_{j*}$

13: for $j = j^* + 1, j^* + 2, ..., q$ **do**

将聚类批次 t_j 重新编号 t_{j-1}

14: end for

删除距离矩阵 M 的第 j^* 行和第 j^* 列

15: for $j = 1, 2, ..., q-1$ **do**

16: $M\left(i^*, j\right) = d\left(t_{i^*}, t_j\right)$

17: $M\left(j, i^*\right) = M\left(i^*, j\right)$

18: end for

19: $q = q - 1$

20: end while

return: 批次划分 $t = \{t_1, t_2, ...t_k\}$

6.2.3 分批次排样优化求解

这里对已经分完的每个批次单独运用问题一所述的改进智能优化算法进行求解，不再过多赘述。

6.2.4 结果展示

表 3 层次聚类下智能优化算法对模型求解的效果

数据集	总批次数	结果指标 【利用率】	总时间
订单 B1	43	70.21%	181.34s
订单 B2	27	68.74%	100.42s
订单 B3	27	69.04%	101.56s
订单 B4	27	70.54%	97.02s
订单 B5	42	68.80%	180.12s

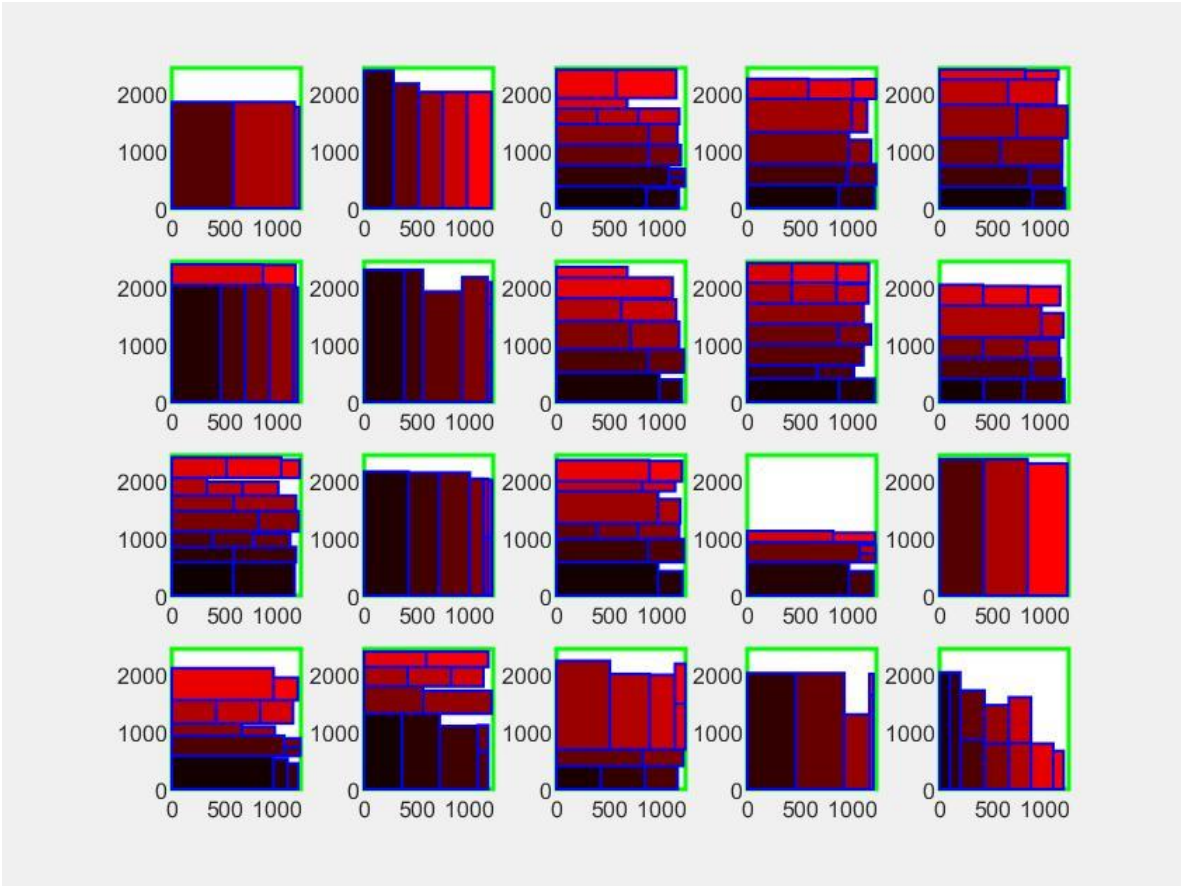

图 19 dataB1 批次一刀切部分排版(任意单批次)

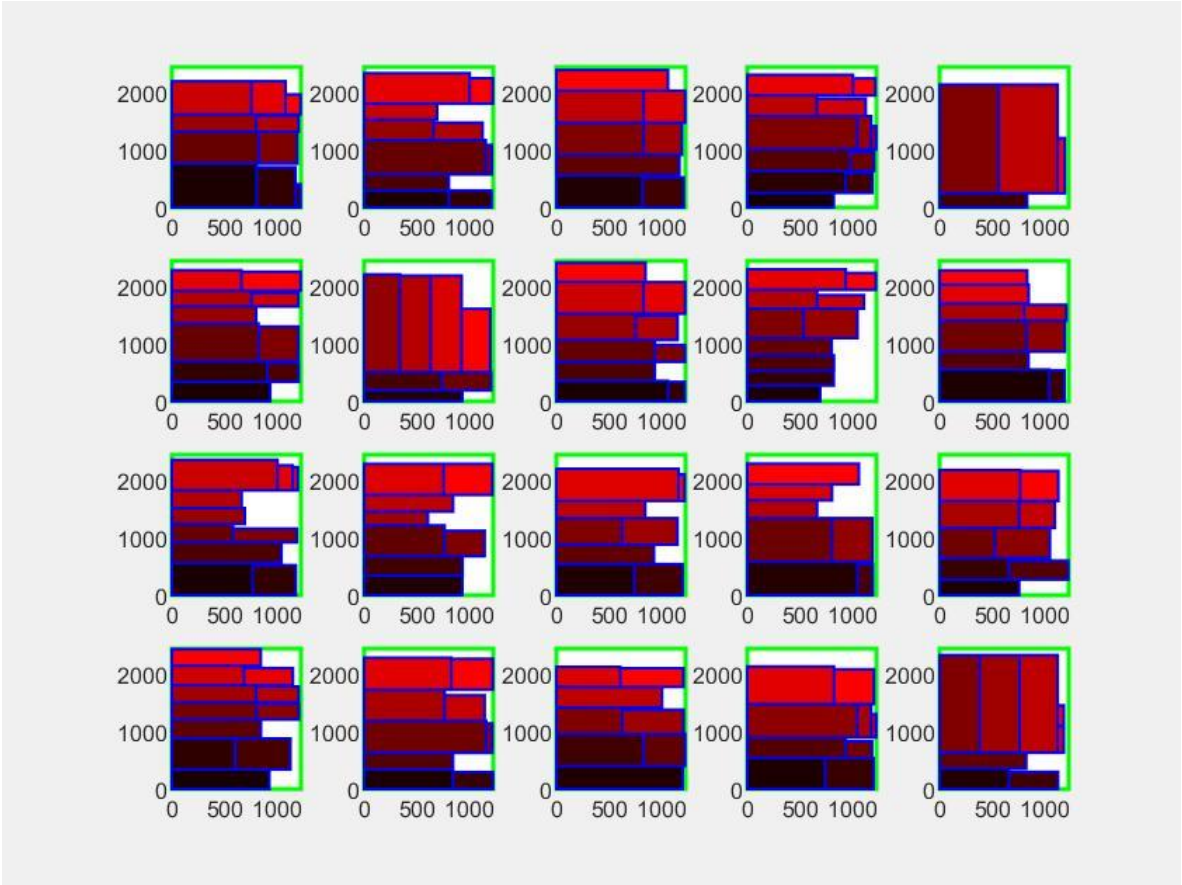

图 20 dataB2 批次一刀切部分排版(任意单批次)

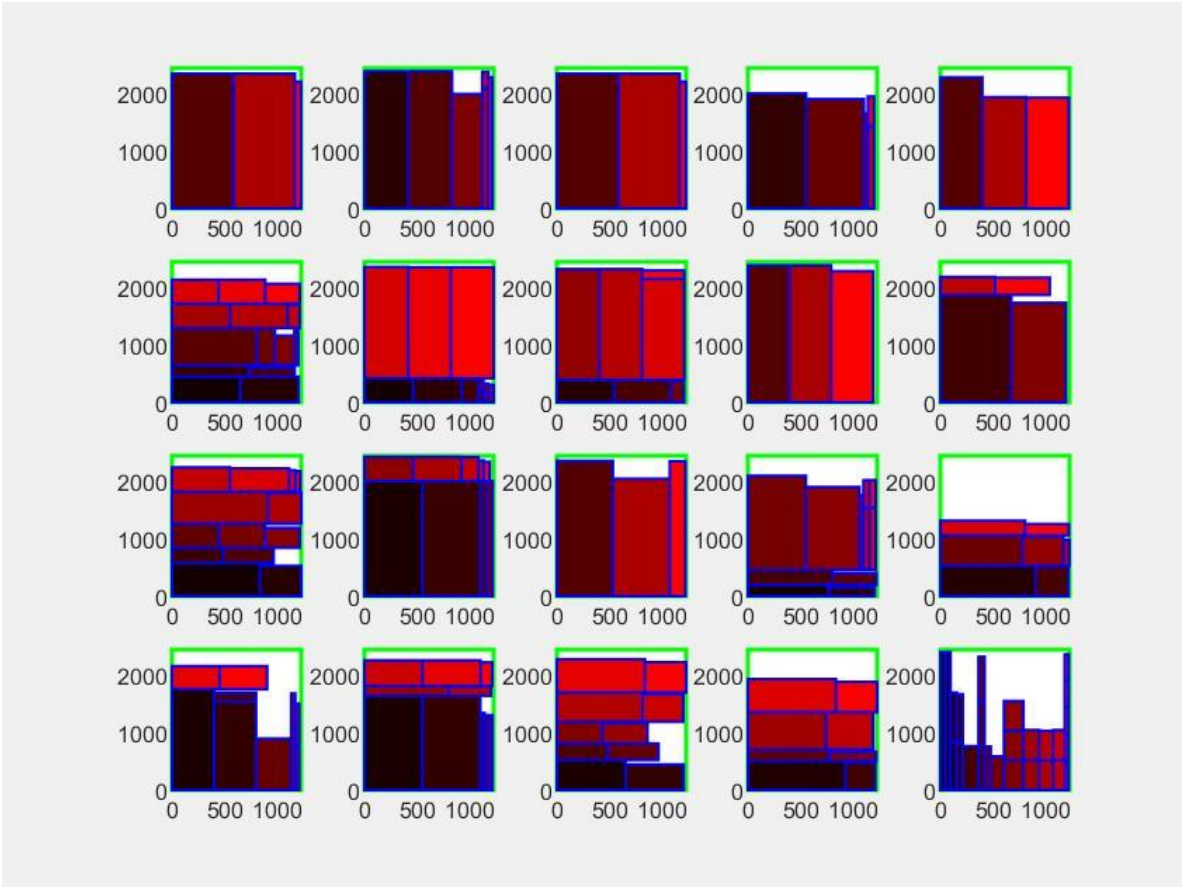

图 21 dataB3 批次一刀切部分排版(任意单批次)

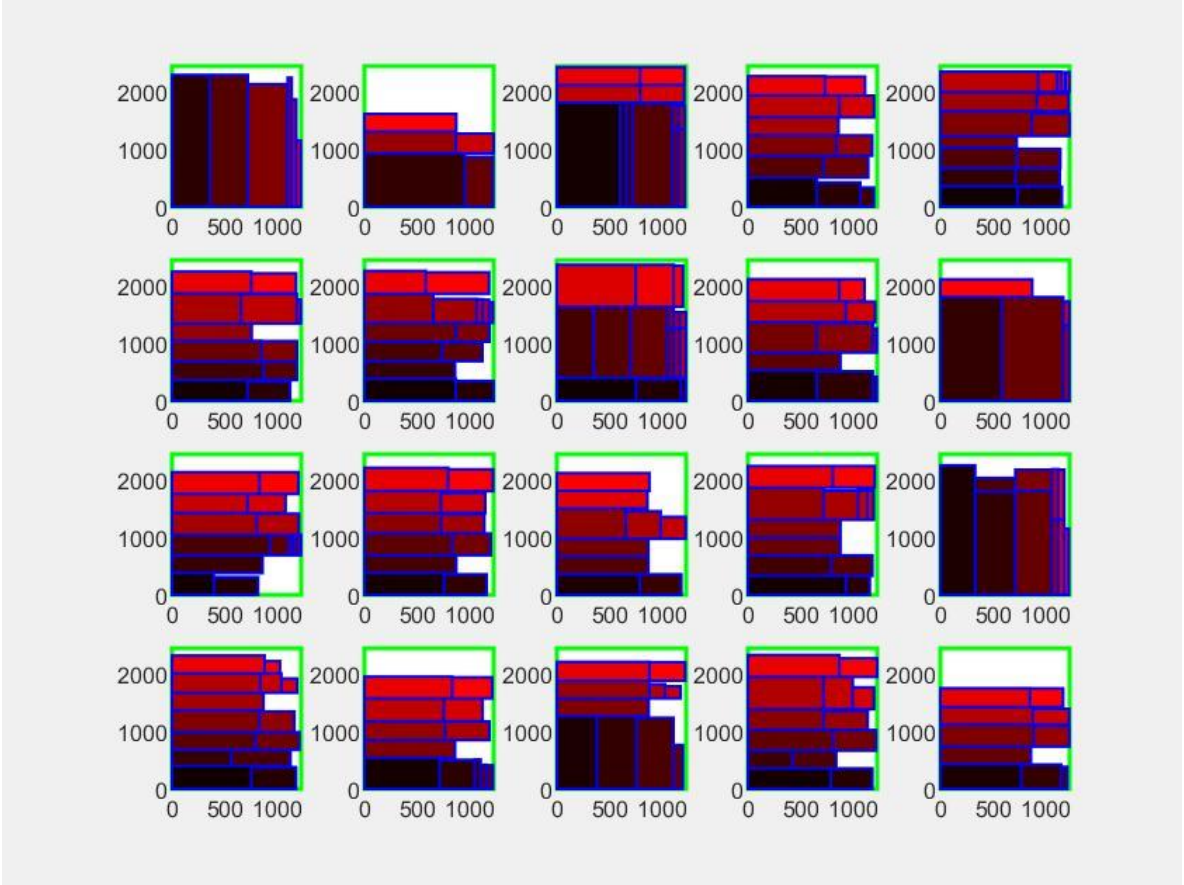

图 22 dataB4 批次一刀切部分排版(任意单批次)

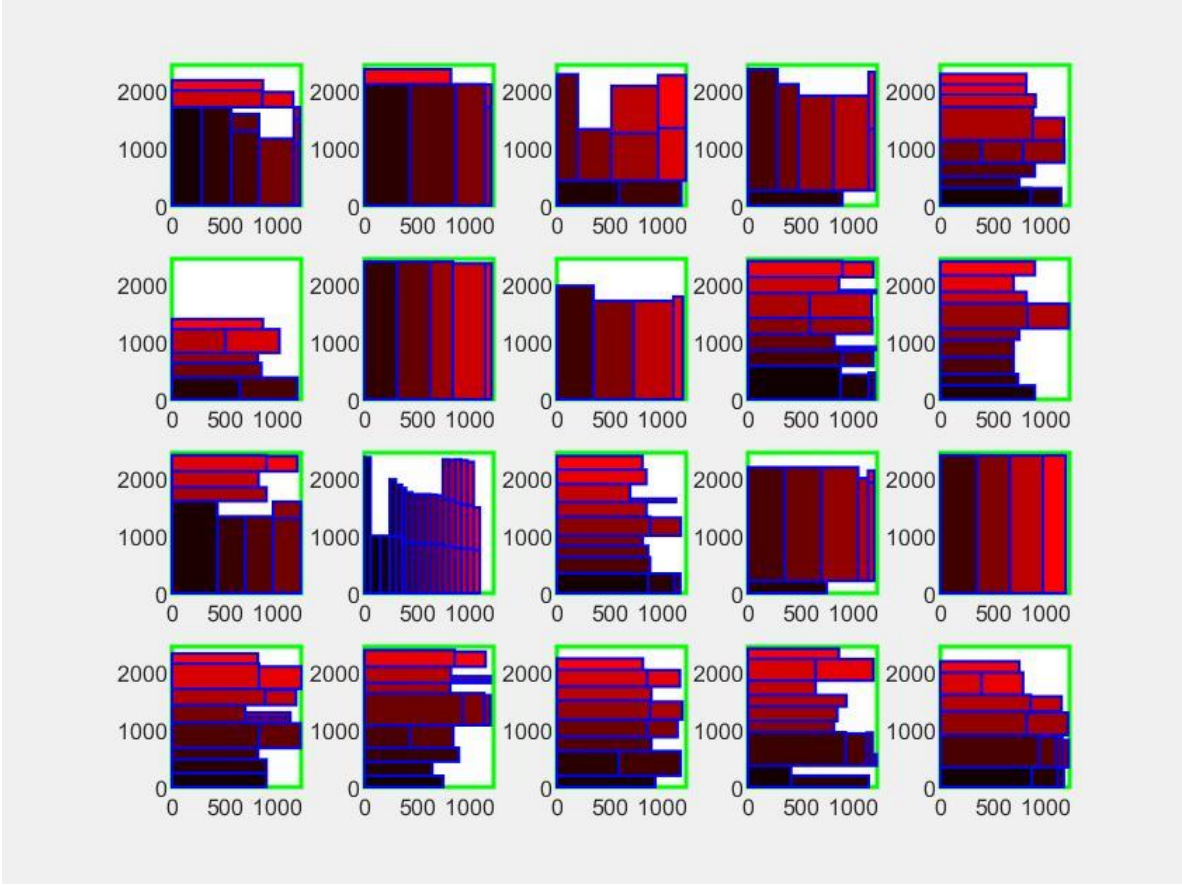

图 23 dataB5 批次一刀切部分排版(任意单批次)

七、 总结与展望

1. 在对问题一的建模求解过程中，粗略考虑到从切割方式角度构建的混合整数规划模型的求解较为复杂，另一方面时间与精力有限并没有对其求精确解。在以后学习中，将进一步研究求出精确解，验证对该模型的复杂的考量。
2. 对问题一的混合整数规划模型我们采取了启发式算法求取近似解，同时考虑到本文数据集的复杂程度，仅仅采取单一的启发式优化算法容易陷入局部最优，于是从板块的排列方式角度引入贪婪搜索算法进行了优化排列，提高了启发式算法的收敛效果，使其避免陷入局部最优。成功的实现了问题一模型的求解同时得到了较高的板材利用率。但是在启发式优化算法的选择和改进上还有研究的空间。例如遗传算法的适应度指标，我们可以通过模拟退火算法进行优化；并且我们还可以使用更多的优化算法比较其收敛效果。
3. 对于问题二，我们以板材消耗最少为优化目标构建了混合整数规划模型，采用层次聚类法对批次进行了划分。其中我们采用了基于评价指标的距离度量进行批次聚类。但其实这种评价指标只是我们人为从题目中发现的隐性条件，主观性过强。为此我们将在以后的研究中寻找一种相对全面且客观的方法对距离度量进行改进。
4 本文所构建的排样优化模型和分批排样优化模型在现实生活中有着很好的推广性，例如地板砖的排样下料问题，钢材的排样下料问题，木板切割等问题。可以很大程度的减少企业的生产成本，时间成本。

八、 参考文献

[1]陈贵敏， 贾建援， 韩琪. 粒子群优化算法的惯性权值递减策略研究[J]. 西安交
通大学学报， 2006， 40(1):5.

[2] Silva E， Alvelos F， Valério de Carvalho J M. An integer programming model for two-and three-stage two-dimensional cutting stock problems [J]. European Journal of Operational Research， 2010， 205(3): 699-708.

[3] Cui Y D， Huang B X. Reducing the number of cuts in generating three-staged cutting patterns [J]. European Journal of Operational Research， 2012， 218: 358-365.

[4] 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.

[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.

附录

附录 1

两问部分代码，用 matlab 语言编写

第一问(部分)：

```
%遗传算法相关
    population_num=200;                %种群大小
    Max_gen= 100;                      %迭代次数
    Pc=0.9;                            %交叉概率
    Pm=0.1;                            %变异概率
    idNum=calcNum;                     %染色体长度==ID 数
    population=zeros(population_num,idNum);
    for i=1:population_num
        population(i,:)=randperm(idNum);
    end
    for i=1:idNum
        if mod(i,2)==0
            population(1,i) = idNum - i/2 + 1;
        else
            population(1,i) = (i+1)/2;
        end
    end
    BEST_VALS = ones(Max_gen,1).*9999;
    %开始循环迭代
    y=1;%循环计数器
    %  计算时间
    tic
    while y<=Max_gen
        fixedPop = population(1: population/10,:);
        %交叉
        [new_pop_intercross]=cross(population_num,population,Pc);
        %变异
        [new_pop_mutation]=mutate(new_pop_intercross,Pm);
        %保留最优
        new_pop_mutation(1: population/10,:) = fixedPop;
        %计算目标函数
       [Result,Groups]= Calculate (new_pop_mutation,stripe,startIDX,plate_length);
        Total_Dis = Result(:,1);

        %更新种群
        new_pop_new=zeros(population_num,idNum);
        [Total_Dissort, index] = sort(Total_Dis);
        for k=1:population_num
            new_pop_new(k,:)=new_pop_mutation(index(k),:);
        end
```

```
        population=new_pop_new;
        %迭代次数加一

        BEST_VALS(y) = Total_Dissort(1);
        BEST_Group = Groups(index(1),:);
        assignin("base","Loss1",BEST_VALS);
        if mod(y,10) == 0
        fprintf('已迭代%d 次\n',y);
        end
        y=y+1;
    end
    toc
    BEST_NUM = BEST_NUMS(Max_gen);
% PSO 算法参数设置
    idNum=calcNum;                  %染色体长度= ID 数
    population_num=200;             %种群大小
    c1 = 2;                         %  加速常数
    c2 = 2;                         %  加速常数
    w = 0.6;                        %  惯性因子
    Vmax = idNum;
    Vmin = 1;
    max = Vmax;
    min = -Vmax;
    Max_gen=100;                        %迭代次数
    VStep=zeros(population_num,idNum);
    for i=1:population_num
        VStep(i,:)=randperm(idNum); %  初始化速度
    end
    assignin("base","VStep",VStep);
    N=idNum;                        %染色体长度=ID 数目

    population=zeros(population_num,idNum);
    for i=1:population_num
        population(i,:)=randperm(idNum);
    end
    for i=1:idNum
        if mod(i,2)==0
            population(1,i) = idNum - i/2 + 1;
        else
            population(1,i) = (i+1)/2;
        end
    end
    BEST_NUMS = ones(Max_gen,1).*9999;
    %开始循环迭代
```

```
y=1;%循环计数器
tic
while y<=Max_gen
    fixedPop = population;
    newPop = population(1:population_num/10,:);
    for j=1:population_num
        % 速度更新
        for m=1:N
        VStep(j,m) = w*VStep(j,m) + c1*rand*randi([min,max]) +
        c2*rand*randi([min,max]);
        end
        if VStep(j,:)>Vmax, VStep(j,:)=Vmax; end %限定范围
        if VStep(j,:)<Vmin, VStep(j,:)=Vmin; end
        % 位置更新
        fixedPop(j,:)=fixedPop(j,:)+VStep(j,:);
        %通过 unique 函数去除数组中相同的元素，然后比较俩个数组的长度
        uniq=N-length(unique(fixedPop(j,:)));
        %如果结果为 0.则说明俩个数组中没有相同的元素
        if uniq~=0
        continue;
        end
        % argsort
        sortPop = sort(fixedPop(j,:));
        for k=1:N
        fixedPop(j,k) = find(sortPop==fixedPop(j,k));
        end
        % 更新
        population(j,:)=fixedPop(j,:);
    end
    %保留最优
    population(1:population_num/10,:) = newPop;
    %计算目标函数
    new_pop_new=zeros(population_num,idNum);
    [Result,Groups]= Calculate (population,stripe,startIDX,plate_length);
    Total_Dis = Result(:,1);
    [Total_Dissort, index] = sort(Total_Dis);
    % 粒子群更新
    for k=1:population_num
        new_pop_new(k,:)=population(index(k),:);
    end
    population=new_pop_new;
    %迭代次数加一
    BEST_VALS(y) = Total_Dissort(1);
    BEST_Group = Groups(index(1),:);
```

```
            assignin("base","Loss2",BEST_VALS);
            if mod(y,10) == 0
            fprintf('已迭代%d 次\n',y);
            end
            y=y+1;
        end
        assignin("base","population",population);
        toc
        BEST_NUM = BEST_NUMS(Max_gen);

%部分优化：
%% 条带生成函数
function stripe=stripe_create(plate_width,plate_length,rectangles)
% 初始化条带记录
stripe={};

result.config=[];
result=repmat(result,1,length(rectangles));

% 初始化产品项，判断产品项的长边，按照先宽后长的顺序保存数据
remaining=rectangles;
for ii=1:size(remaining,1)
        if (remaining(ii,1)>remaining(ii,2))
            remaining(ii,:)=remaining(ii,[2 1]);
        end
end

% 将产品项按面积从大到小排序
[~,sorted_indices]                                                                    =
sort(remaining(:,1).*remaining(:,2),'descend');sorted_indices=sorted_indices';
remainingS=remaining(sorted_indices,:);

while ~isempty(sorted_indices)

        % 初始化已排样产品项记录
        already_sampling=[];

        % 初始化条带的左下角坐标
        x= 0;y=0;

        % 初始化条带剩余可用宽度和高度（板材竖着放，所以条带是宽和高）
        w=0;h=0;

        % 初始化当前板材被横切掉的高（板材竖着放）
```

```
    H = 0;

    % 在所有的剩余产品项 remaining 中寻找（产品项长>板材宽，且产品项长+被切掉的板材的高<板材高）
    % 或者（产品项长<板材宽，且产品项宽+被切掉的板材的高<板材高）的面积最大的产品项
    % 即在板材竖着放的情况下，寻找可以横放或竖放的面积最大的产品项
    Ind0=find((remaining(sorted_indices,2)>plate_width                    &
                        H+remaining(sorted_indices,2)<plate_length)
        |                  (remaining(sorted_indices,2)<plate_width       &
                            H+remaining(sorted_indices,1)<plate_length),1);
    if (isempty(Ind0))
        break;
    end

    % 获取满足上述条件的最大的产品项
    idx = sorted_indices(Ind0);

    % 最大的产品项的宽和长，r(2)是长，r(1)是宽
    r = remaining(idx,:);
    if (r(2)>plate_width)   % 产品项长>板材宽
            result(idx).config = [x, y, r(1), r(2)]; % 将产品项竖排，记录产品项左下角坐标和，宽，长数据
            % 更新条带可用部分的左下角坐标
            x= r(1);y=H;
            % 更新条带剩余可用宽度和高度（板材竖着放，所以条带是宽和高）
            w=plate_width - r(1);h=r(2);
            % 更新板材被横切掉的高（板材竖着放）
            H = H + r(2);
    else % 产品项长<=板材宽
            result(idx).config = [x, y, r(2), r(1)]; % 将产品项横排，记录产品项左下角坐标和，长，宽数据
            % 更新条带可用部分的左下角坐标
            x= r(2);y=H;
            % 更新条带剩余可用宽度和高度（板材竖着放，所以条带是宽和高）
            w=plate_width - r(2);h=r(1);
            % 更新板材被横切掉的高（板材竖着放）
            H = H + r(1);
    end

    % 记录已排样产品项
    already_sampling=[already_sampling;sorted_indices(Ind0)];

    % 删除已完成排版的产品项
```

```
        sorted_indices(Ind0)=[];

        %% 调用条带剩余部分内部排样函数
        % 在条带内寻找可以放置的产品项，直到无法再放置新产品项，并计算相关数据
        [sorted_indices,result,already_sampling]=sampling_in_stripe(x, y, w, h, 0, remaining, sorted_indices, result,0,0, already_sampling);

        % 记录现有条带
        stripe(end+1,1)={result(already_sampling)};
        stripe(end,3)={already_sampling};
        stripe(end,2)={H};
        % 在新的板材中生成条带，直到所有产品项均完成排样
        x=0;y=0;

end
```

适应度计算：

```
function [Result,Groups]= Calculate (Total,stripe,startIDX,plate_length)
Result = zeros(size(Total,1),1);
num = size(Total,2);
Groups = cell(size(Total,1),num);
for idx = 1:size(Total,1)
        current = Total(idx,:);
        score = 1;
        total_len = 0;
        for i=1:num
                idx2 = current(i) + startIDX;
                len = stripe{idx2,2};
                if total_len + len > plate_length
                        score = score + 1;
                        total_len = len;

                else
                        total_len = total_len + len;
                end
                Groups{idx,score} = [Groups{idx,score},idx2];
        end

        Result(idx) = score;
end
end

function result = SortStruct(struct,idx)
```

```
planToSort = zeros(size(struct,idx),1);
    for i=1:size(struct,1)
        planToSort(i) = struct{i,2};
    end
     [~,index]=sort(planToSort,'descend');
     result=struct(index,:);
end
```

第二问(部分)：

```
%按批次做层次聚类
function result= Batch clustering(Rectangle,t_width,t_length)

result.config=[];
result=repmat(result,1,length(Rectangle));
rect_st=Rectangle;
for ii=1:size(rect_st,1)
    if (rect_st(ii,1)>rect_st(ii,2))
        rect_st(ii,:)=rect_st(ii,[2 1]);
    end
end

[~,idx2] = sort(rect_st(:,1).*rect_st(:,2),'descend');
idx2=idx2';
x= 0;y=0;w=0;h=0;H = 0;

while ~isempty(idx2)
    Ind0=find((rect_st(idx2,2)>t_width & H+rect_st(idx2,2)<t_length) ...
        | (rect_st(idx2,2)<t_width & H+rect_st(idx2,1)<t_length),1);
    if (isempty(Ind0))
        break;
    end
    idx = idx2(Ind0);

    idx2(Ind0)=[];
    r = rect_st(idx,:);
    if (r(2)>t_width)
            result(idx).config = [x, y, r(1), r(2)];
            x= r(1);y=H;w=t_width - r(1);h=r(2);H = H + r(2);
    else
            result(idx).config = [x, y, r(2), r(1)];
            x= r(2);y=H;w=t_width - r(2);h=r(1);H = H + r(1);
    end
    [idx2,result]=MyArrangeFunc2(x, y, w, h, 0, rect_st, idx2, result,0,0);

    x=0;y=H;
```

```
end
function [indices,result]=MyArrangeFunc2(x, y, w, h, D, rm, indices, result,dc,dgw)

dc=dc+1;
p1 = 6.1;

for idx=indices
    if (dc==1)
        for j =1:D + 2
            if p1 > 1 && rm(idx,mod(j-1,2)+1) == w && rm(idx,mod(j,2)+1)    == h
                p1=1;
                orientation= j;
                best =idx;
                break;
            elseif p1 > 2 && rm(idx,mod(j-1,2)+1) < w && rm(idx,mod(j,2)+1)    ==
h
                p1=2;
                orientation= j;
                best =idx;
            elseif p1 > 3 && rm(idx,mod(j-1,2)+1) == w && rm(idx,mod(j,2)+1)    >
0.75*h && rm(idx,mod(j,2)+1)    <h
                p1=3;
                orientation= j;
                best =idx;
            elseif p1 >= 4 && rm(idx,mod(j-1,2)+1) <= w && rm(idx,mod(j,2)+1)    <
h
                whz=((rm(idx,mod(j-
1,2)+1)/w*rm(idx,mod(j,2)+1)/h)+rm(idx,mod(j,2)+1)/h)/2;
                if (5-whz<p1)
                    p1=5-whz;
                    orientation= j;
                    best =idx;
                end
            elseif p1 > 5
                p1=5;
                orientation= j;
                best =idx;
            end
        end
    else

        for j =1:D + 2
            if p1 > 1 && rm(idx,mod(j-1,2)+1) == dgw && dgw==w    &&
rm(idx,mod(j,2)+1)    == h
```

```
                    p1=1;
                    orientation= j;
                    best =idx;
                    break;
                elseif p1 > 3 && rm(idx,mod(j-1,2)+1) == dgw && dgw==w  &&
rm(idx,mod(j,2)+1)  < h
                    p1=3;
                    orientation= j;
                    best =idx;
                 elseif p1 > 2 && rm(idx,mod(j-1,2)+1) ==dgw && rm(idx,mod(j,2)+1)
== h
                    p1=2;
                    orientation= j;
                    best =idx;
                elseif p1 > 4 && rm(idx,mod(j-1,2)+1)==dgw && rm(idx,mod(j,2)+1)  <
h
                    p1=4;
                    orientation= j;
                    best =idx;
                elseif p1 > 5
                    p1=5;
                    orientation= j;
                    best =idx;
                end
            end
        end
end

if (p1<5)
        if orientation == 1
            omega=rm(best,1); d=rm(best,2);
        else
            omega=rm(best,2); d=rm(best,1);
        end
        result(best).config = [x, y, omega, d];
        if (dc==1)
            dgw=omega;
        end
        indices(indices==best)=[];
        if p1 == 3
            [indices,result]=MyArrangeFunc2(x, y + d, w, h - d, D, rm, indices,
result,dc,dgw);
        elseif p1 == 2
            [indices,result]=MyArrangeFunc2(x + omega, y, w - omega, h, D, rm, indices,
```

```
result,0,0);
    elseif p1 >=4 && p1<5
        min_w = inf;
        min_h = inf;
        for idx=indices
            min_w = min(min_w, rm(idx,1));
            min_h = min(min_h, rm(idx,2));
        end
        min_w = min(min_h, min_w);
        min_h = min_w;
        if w - omega < min_w
            [indices,result]=MyArrangeFunc2(x, y + d, w, h - d, D, rm, indices,
result,dc,dgw);
        elseif h - d < min_h
            [indices,result]=MyArrangeFunc2(x + omega, y, w - omega, h, D, rm,
indices, result,0,0);
        elseif omega < min_w
            [indices,result]=MyArrangeFunc2(x + omega, y, w - omega, d, D, rm,
indices, result,dc,dgw);
            [indices,result]=MyArrangeFunc2(x, y + d, w, h - d, D, rm, indices,
result,dc,dgw);
        else
            [indices,result]=MyArrangeFunc2(x, y + d, omega, h - d, D, rm, indices,
result,dc,dgw);
            [indices,result]=MyArrangeFunc2(x + omega, y, w - omega, h, D, rm,
indices, result,0,0);
        end
    end
end
```