


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	电子科技大学
参赛队号	22106140009
队员姓名	1.彭松林
	2.邬云舒
	3.刘玉琴

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　二维三阶段矩形件排样组批优化研究

摘　　要

“个性化定制”成为企业在智能制造转型中的主要着力点，其中方形件产品的个性化定制通过设定订单组批来实现批量切割，以提高原料利用率。在定制化生产模式中，订单组批和排样优化至关重要，如果组批批次太小，原料利用率底，生产效率底；如果组批批次太大，原料利用率会提高，但不能保证准时交货，可见个性化与生产高效之间存在极大矛盾。对下料的排样优化，合理规划方形件的布局，能减少下料阶段的原料浪费，简化切割过程。因此，如何优化方形件产品的布局，如何将订单分组分批，是提高材料利用率、节约资源和能源、承担环境责任所要解决的关键问题。

排样优化问题（问题一），根据题目要求，首先根据**一刀切、最多 3 个阶段的切割、精确切割、原片长宽固定等约束条件**建立了求解最大利用率（最少原片数）的**混合整数规划模型**。给定数据里面存在很多长度或宽度相等的产品项，我们**构造二元组块的实用性算法（2-items）**进行两两组合，用新生成的数据集 Item_list 替代数据集 dataA。题目要求同一个栈里的产品项长或宽应该相同，我们再对 Item_list 数据里具有相等边的单元合成一个完整的栈，并建立 stack_list 数据集，再以栈为最小单元的进行排样。题目要求三阶段精确切割，栈为切割的最后一段，以**栈为最小单位**，认为已经完成了最后一段切割，把**三段切割方法转化为两段式切割**方法，减小了排样的复杂度。对后续排样，我们设定从原片左下角开始排样，用栈沿着原片底边方向开始摆放，直到形成一个完整的条带，再**用条带构成整个原片排样**，将问题**转化为条带装箱问题**。区别于恒定宽度的普通条带模型，我们的条带模型的宽度是根据条带里面栈的宽度变化的，因此**提出改进条带模型，**并构造了基于最大适应度优先排布的**启发式条带生成算法（Improved-Stripe）**，最后在边界条件约束下，用条带对每块原片进行排样。最终得到数据集 **dataA1** 需要原片 **89 片**，利用率 **93.87%**；**dataA2** 需要原片 **89 片**，利用率 **93.12%**；**dataA3** 需要原片 **88 片**，利用率 **95.15%**；**dataA4** 需要原片 **87 片**，利用率 **94.08%**。最后对模型和算法进行分析，我们的算法可行性高、实现较为容易、复杂度为线性阶，计算 4 个数据集总计运行时间为 3.9748 秒。

订单组批问题（问题二），根据题目要求，在问题一的基础上添加**订单不能分**

割、不同材料分开切割、同一批次对产品项总数和总面积有限制等约束条件,建立了**求解最大原片利用率的混合整数规划模型**，同时提出**相似度**的概念，通过科学分析影响两个订单间相似度的因素，找到 3 个指标M^1，M^2，M^3，并利用**变异系数法**求得 3 个指标的**权重分别为** 0.4780，0.4955，0.0265，得到了公式 6-2 所示的**相似度计算公式**。以相似度作为依据提出了**基于相似度聚类的订单分批排样算法**，该算法在满足约束条件的前提下，**合并相似度较高的两个订单，不断生成更大的新订单**，在递归的思路基础上，把订单合并到不同批次里，直至生成全部批次。对生成的批次再**按照材料去分类**，对**同一批次同种材料的所有产品项进行排样设计**，得到优秀的排样结果。最终得到数据集 **dataB1** 分为 **42** 个批次，一共需要原片 **3462 片**，利用率 **86.03%**；**dataB2** 分为 **12** 个批次，一共需要原片 **2270 片**，利用率 **84.88%**；**dataB3** 分为 **27** 个批次，一共需要原片 **2298 片**，利用率 **84.15%**；**dataB4** 分为 **30** 个批次，一共需要原片 **2392 片**，利用率 **84.87%**；**dataB5** 分为 **45** 个批次，一共需要原片 **3733 片**，利用率 **82.77%**。最后对模型和算法进行分析和评价，本文科学地提出相似度的概念，灵活运用聚类和递归的思想提出组批算法，对处理订单很多、材料种类很多的大的数据集效果明显，利用率达到预期效果，同时连续计算完成 5 组数据需要花费 156 秒。

最后，结果指标已在正文中以数据和图表的形式呈现，所有计算结果（批次号、原片序号、相对于原点的横纵坐标等）都按照问题要求格式提交，所有主程序和数据结构已在附录中说明。

关键词：排样，混合整数，启发式，变异系数，聚类

目录

1. 问题背景与问题重述

1.1 问题背景

我国制造业正全力向智能制造发展，个性定制化现已是企业智能制造转型中的一个主要竞争点。在离散行业的产品中，例如电子元件、汽车零件和航天领域的零部件等，这些都是依赖于分散生产、加工和灵活组装的部件。对于这一类产品，客户方对于产品的需求是兼具批量化与个性化的，且有极高的质量要求。因此，“个性定制化”服务在市场中有着极具广泛的需要，但同时也增大了企业的需求分析、产品设计和加工的能力，同时也要求企业具有精细的生产流程和完善产品的设计体系。

以板材为主要原料的方形件产品，如通讯、计算机，芯片、板式家具、玻璃等行业的产品，大多采用“品类多、批量小”的个性化定制生产，因为行业的需求量大，生产厂商通常采用“订单组批+批量生产+订单分拣”的模式进行生产，通过对订单设置批组以来完成批量切割，从而提高对于原材料的利用效率。

在定制化生产模式中，订单的批组与组件的排样优化起着至关重要的作用，订单批组把不相同的订单组合成若干个批次，达到生产的批量化。但对于小批量、多品种和大规模的订单进行组批生产时，批次太小，材料的利用率不高，而且生产效率还低；对于太大的生产批次，虽然可能提高材料的利用率，但会影响交货时间，这就体现了个性化与生产效率之间的矛盾。

方形件产品的排样优化实际上是一个下料优化问题，优化的目的是更好的规划方形件的布局，减少原料在切割过程中的浪费，这种问题是高复杂度的组合优化问题，也是运筹学的一个分支。在下料过程中，如何提高原材料的利用率，减少废料，是节约资源和能源所要解决的关键问题。

1.2 问题重述

需要考虑交货期、设备生产能力、仓库存储量、原料利用率、生产效率等因素来进行组批优化。把材质相同、交货时间相近的订单放在一个批次里，用组批优化来保证订单期限，提高原料的利用率和产品的生产效率。要求每个批次必须完整的完成一个或多个订单，且订单的交货期都相同。

排样优化指：选着方形件的数量和大小，同时定下原片的数量，使板材的利用率最大化。切割时需要对每片板材进行“一刀切”，即每次切割都要保证板材被切割成两块。同时还要求采用三段式、精确的切割方式，三段式指的是切割方向只能最多变换两次，且每次还必须满足“一刀切”要求；精确切割指三个阶段内就能切割出准确尺寸的方形件（具体可参见图 1-1）。

以图 1-1 为例，对三阶段、精确、一刀切进行说明。中间红线为第一次切割，将板材七个为上下两块；再将两块板材都顺时针旋转 90°，按着蓝色线条进行切割，此次切割为第二段；最后第三段切割，依照绿色的线，把每块组件切割下来，三段切割完成后，已经将计划好的方形组件全部切割好，且不需要第四段切割。

组批问题指：一个交货期内有多个订单需要加工，每个订单的材质和工件的样式各不相同，每个需加工工件的数量多少不一，但为了保证生产设备保持高产能负荷，就必须把不满足产能负荷的订单进行合并，这就是组批问题。但一个数据里面穿在几百个不同的订单，如何组合订单使材料利用率最大化是本题需要解决的问

题。进行组批的条件约束条件有订单、材质、item 数量、item 总面积，协调这几个约束条件，使得组批后的利用率最大化。

1.3 问题要求

问题 1：排样优化问题

在约束条件：在相同栈（stack）里的产品项（item）的宽度（或长度）应该相同；最终切割生成的产品项是完整的，非拼接而成。在约束条件下，以数据集 A 为输入数据，要求要求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。并得出每个方形组件在板材上的位置和相应的板材标号，并给出原材料的利用率。

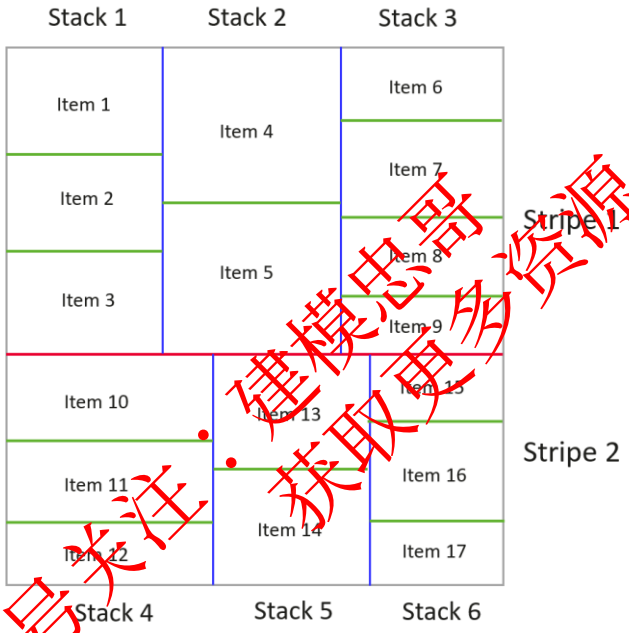

图 1-1 不同切割阶段的形式定义

问题 2：订单组批问题

在约束条件：满足问题一的切割条件；每个订单只能一个批次完成；批次的 item 总数不能超过 1000 个，总面积不能超过 250mm^2，在这几个要求下对每个数据集进行组批，组批后进行排样，再求每个数据的原材料利用率。

2. 基本假设与符号系统

2.1 模型假设

假设 1. 假定板材原片仅有一种规格且数量充足；
假设 2. 排样方案不用考虑锯缝宽度（即切割的缝隙宽度）影响；
假设 3. 假设每次切割必须成功；
假设 4. 假设所给数据真实有效；
假设 5. 假设生产条件足够架构完所有工件；
假设 6. 假设所有订单都为统一交货时间，且时间充裕；

2.2 符号说明

符号	说明	量纲
E_i	第 i 块原片的利用率	%
L	原片长度	mm
W	原片宽度	mm
i	第 i 个组件	--
l_i	第 i 个组件的长度	mm
w_i	第 i 个组件的宽度	mm
$F1i$	第一适应度	%
$F2i$	第二适应度	%
S_i	表示第 i 个订单面积	m^2
x_i	表示第 i 个订单	--
I	表示总的组件的个数	--

3. 问题一的分析

3.1 数据预处理

首先对用程序对所给数据的数值信息提起，查找是否存在数据缺失或者错误数据，结果不存在数据遗漏或者缺失。用每个方形件产品的长度都与原片长度（2240 毫米）进行比较，未发现大于原片长度的错误数据，然后对方形件产品的宽度都与原片长度（1220 毫米）进行比较，也未发现大于原片宽度的错误数据，说明所给的所有方形件产品数据都是有效数据。

3.2 问题一分析

在题目的总体要求（一刀切、三段切割、精确排样、板材原片规格统一、不考虑切割损耗）下，可以把任务要求归纳为：按照切割要求，把订单中设计好的大小各异的方形件产品准确地切割出来，并且把全部产品切割出来。在实际情况中，每批订单的签约和交货期不一定都相同，旺季订单可能堆积，淡季订单可能不足，且交货时间也不统一。题目要求把订单进行合理的规划加工批次，以来保证材料利用率、生产效率和交货期。但题目考虑到于统一处理数据和体现问题本质，所有订单都为同一期交货，这样简化了分析过程，现在只需在所有订单中，按照每批次的生产上限，设计本批次的组批。因此，组批优化问题是本题所要解决的关键问题，也是分析问题的重点。

在下料过程中，每块材料原片上的方形件排布、尺寸、数量都对板材原片的利用率有着很大影响，所有材料原片的利用率的总和构成生产这批订单总的利用率。因此，需要在每片原材上着手，最大化板材原片利用率。排样优化也成为了本题和核心问题。

根据题目一的要求，题目将上述两个问题分步解决，在问题一中只需要先解决排样优化问题，达到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的要求。

在约束条件（一刀切、三段切割、精确排样、相同 stack 里的 item 的宽度或长度应该相同）下的切割示意图，如图 3-1 所示。

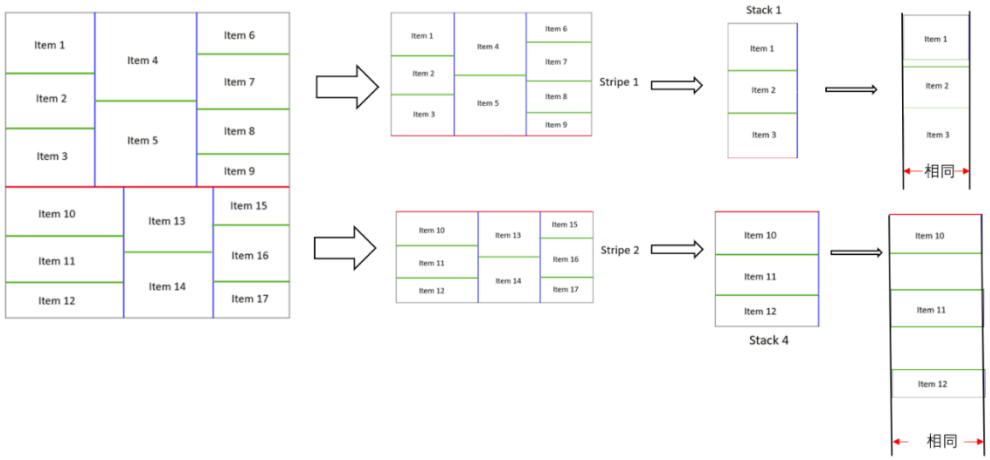

图 3-1 切割示意图

从示意图中看出，可以把单边相同的 item 进行合并，这样能减小数据量，而且大面积的产品块更利于排样优化。题目要求在相同 stack 里的 tem 的宽度（或长度）相同，这一点更论证了合并的可行性。在合并了组元之后，用合并之后的数据进行排样，这样计算难度和排样难度会相对减小。

因为第一题不需要考虑组批问题，只需要把每个数据集里面的工件排样完成即可，要求利用率最大化。板材的利用率由单块板材利用率累加而成，单块板材的利用率又由板材上组件的排样决定，最终归结于寻找出最优化的排样方案。问题一的求解思路流程如图 3-2 所示。

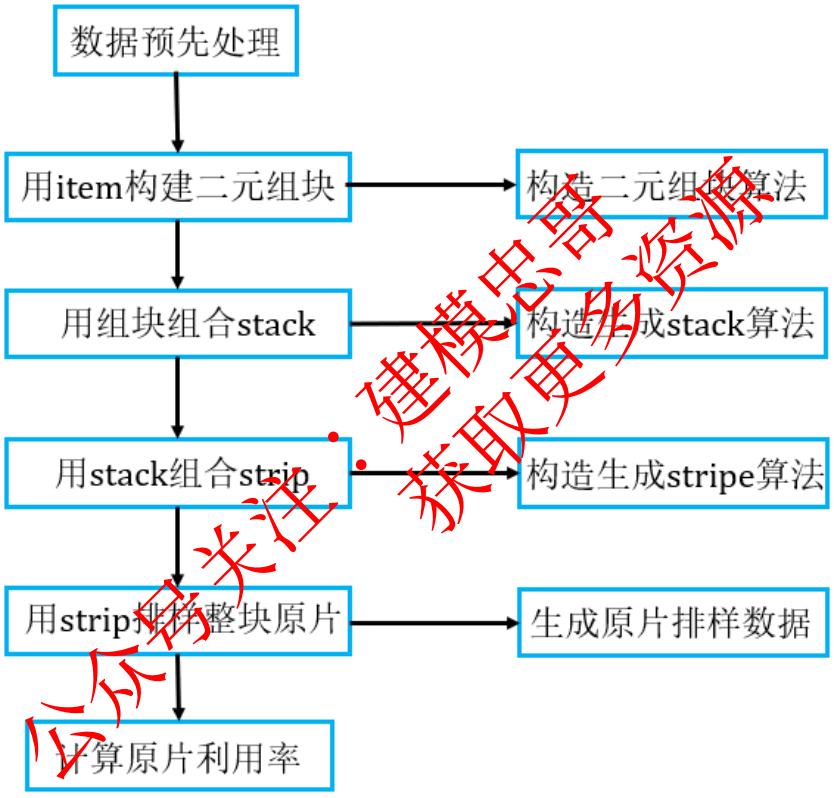

图 3-2 切割示意图

3.3 问题一的模型建立

3.3.1 问题一求解目标

问题一最终问题归结于最大化排样的利用率，以利用率为目标函数建立混合整数规划模型，目标函数为：

$$\max E = \frac{\sum_i^I l_i w_i n_i}{N \times L \times W} \qquad (3\text{-}1)$$

其中 E 表示原片（板材）利用率（efficiency），L 表示原片长度，W 表示原片宽度，I 表示待排样 item 的种类数，i 代表第 i 个 item，l_i 表示第 i 个 item 的长度，w_j 表示第 i 个 item 的宽度，n_i 表示原片上排布的 i 类 item 的数量,N表示需要的原片总数[1]。

以板材原片的左下角为坐标原点，长和宽分别为坐标系的 x 与 y 轴建立笛卡尔坐标系，以 item 的左下角为每个 item 的起始点。排样的约束条件模型表示为：

（1）item 的总长度不能超过板材的长度

$$\sum_{j=1}^{N} l_j \le L \quad j = 1, \dots, N \tag{3-2}$$

（2）item 的总宽度不能超过板材的宽度

$$\sum_{j=1}^{N} w_j \le W \quad j = 1, \dots, N \tag{3-3}$$

（3）余料的长度要小于最短 item 的长度

$$L - \sum_{j=1}^{N} l_i \le \min\{l_1, l_2, l_3, \dots, , l_N\} \quad j = 1, \dots, N \tag{3-4}$$

以利用率为目标函数建立混合整数规划模型为：

$$maxE = \frac{\sum_i^I l_i w_i n_i}{L \times W}$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{N} l_j \le L \\ \sum_{j=1}^{N} w_j \le W \\ L - \sum_{j=1}^{N} l_i \le \min\{l_1, l_2, l_3, \dots, , l_N\} \end{cases}$$

3.3.2 组块的前置处理

题目要求每个相同 stack 里的 item 的宽度（或长度）应该相同，从此角度出发，应该把具有相同特征参数的组块合并在一起（组块合成），需要将数据进行前置处理。但所给数据中具有相同参数的组块单元（单元组块 item）能有两个或两个以上，考虑到两个元组块合并成的组合（二元组块 2-items）还能接着再进行合并，决定以 2-items 的组合为够着的基础单元，组合过程如图 3-2 所示。

2-items 构造的条件：

（1） 被组合块具有相同边（宽度或长度相等）；

（2） 被组合 item 的不相等边之和不大于板材元件的最大边；

（3） 组合后的 2-items 应当具有两块单元组块的标号。

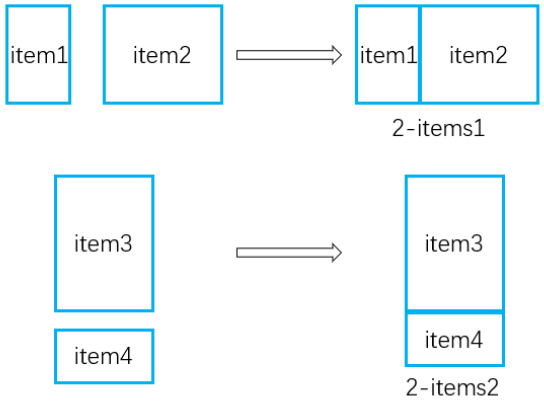

图 3-2 组块组合示意图

3.3.3 组件的放置规则

要把每个 Group_list 中的 item 和 item2 排布在材料原片上，首先需要制定好合理的排布规则。设定以板材原片的左下角为坐标原点，长和宽分别为坐标系的 x 与 y 轴建立笛卡尔坐标系，这样就能通过坐标将每个 item 准确定位，且更好的计算 item 所占面积，排放规则如图 3-3 所示。

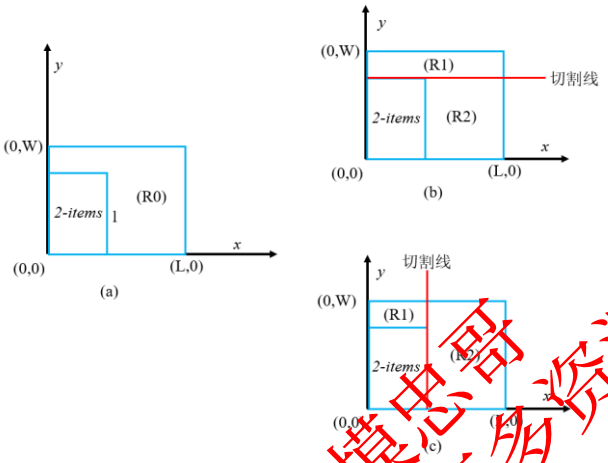

图 3- 3　排放规则和切割方向示意图

如图 3- 所示，将组合件 2-items,放置于左下角，按照一刀切的原则，可有两种切割方法，分别是平行于 2-items 的 w 边，水平切割成上下两半，如图（b）所示；另一种切割方法是沿着 2-items 的 l 方向竖直切割，切成左右两半，如图（c）所示。

在上述两种切割方向中，选定水平切割方式为第一段切割，如图 3-4 所示。第一段将原材料片切割为上下两块，此处只列举了下面部分，上面区域与下同。将 Group_list 数据中的 item 和 2-items 按照图 3- 方式放置，在 2-items-1、2-items-2、item-1 中 2-items-1 最高，则以 2-items-1 的上边沿为切割线，将原板材切割成 R1 区域和下边区域两块，此时下边区域已经被切割为普通条带，其中灰色区域为废料区，这就将矩形装箱问题转化为普通条带装箱问题[2]。

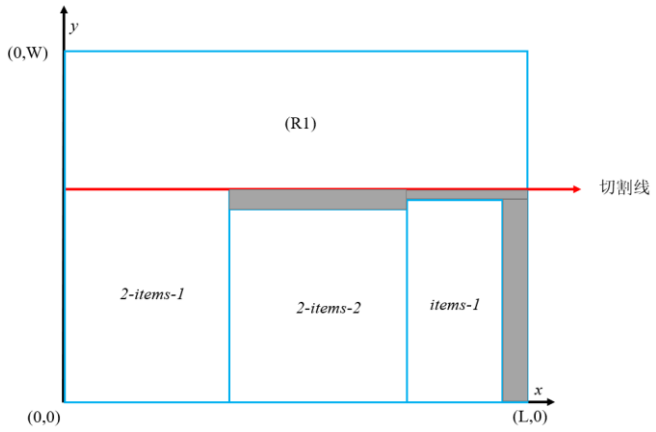

图 3- 4　第一段切割方向示意图

3.3.4 改进条带模型建立

根据设定的 item 放置规则，可以将矩形原片在第一阶段划分成若干个普通条形，这将使矩形装箱问题简化为普通条带装箱问题，同时将模型简化为二阶段普通条带模型，但相比于等宽的条带，此条带模型的宽需要根据最大 item 的宽改变。因此，需要在普通条带模型的基础上进行模型改进。

在解决普通条带模型时，通常使用的是背包算法，将两阶段条带排样的构造转化为用预先生成好的 stack 拼接。stack 拼接需要满足拼接后边长不超过板材的长，且拼接单元需要有相等的边，满足这两个条件才能拼接成一个 stack，stack 拼接示意图如图 3-5 所示。

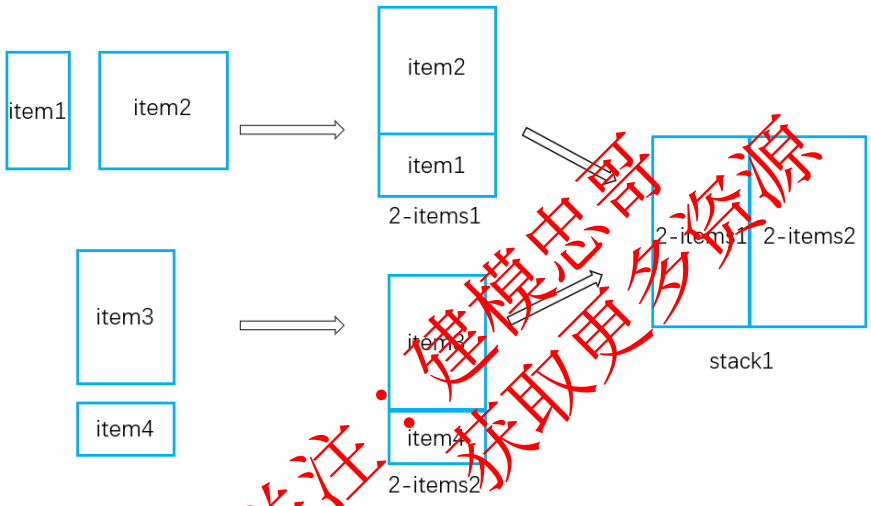

图 3-5　stack 示意图

普通条带模型的所有条带都是同一规格，而本题的条带宽度根据 stack 的宽度改变。因此，需要在普通条带模型的基础上做出相应改进，同时需要对背包算法的约束条进行改进，改进普通条带模型建立如下：

$$maxE_j = \frac{\sum_i^I l_i w_i}{L \times W_j} \quad (i,\ j = 1,2,\dots,I) \qquad (3\text{-}5)$$

其中 j 表示第 j 个条带，E_j 表示第 j 个条带的利用率，L 表示第条带的长（所有条带的长为板材的长），W_j 表示第 j 个条带的宽，l_i 表示第 i 个 Stack 的长度，w_i 表示第 i 个 Stack 的宽度。

改进条带的约束条件模型表示为：

（1）Stack 的总长度不能超过板材的长度

$$\sum_{j=1}^{N} l_i a_{i,j} \le L \quad j = 1,\dots,N \qquad (3\text{-}6)$$

（2）Stack 的总宽度不能超过板材的宽度

$$\sum_{j=1}^{N} w_i a_{i,j} \le W \quad j = 1,\dots,N \qquad (3\text{-}7)$$

（3）余料的长度要小于最短 Stack 的长度

$$L-\sum_{j=1}^{N} l_i a_{i,j} \le min\ \{l_1, l_2, l_3, \dots, , l_N\}\ j=1, \dots, N \qquad (3\text{-}8)$$

在改进后的条带由宽度不同的 Stake 拼接生成，条带的宽度由最左边 Stack 的宽度决定，从左到右 stack 的宽度呈递减趋势，直到所剩区域无法放下任意一个 item，其中 stack_list 数据集合的生成方式和 item_list 生成方式，如图 3-6 所示。

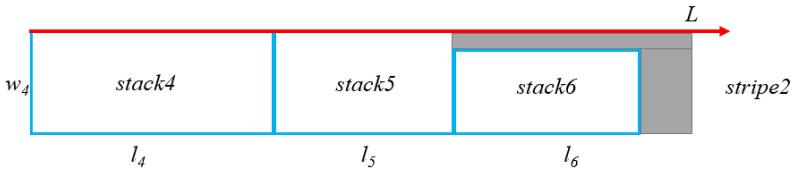

图 3-6　改进条带示意图

每个 stripe 中 stack 的选择决定着整体的利用率，因此需要将问题关键归结为“如何选取每个 stack”，以达到原料最大利用率。构造基于最优适应度评价机制的启发式算法，以面积为适应度条件，在选择第一个 stack 时，首先选择适应度最大的，因为最大适应度的放置在最后，没有太多的小面积与其搭配，可能造成大面积的但排一个原板，这样导致原料利用率低，因此需要首先排适应度大的。选完第一个 stack 之后，搜索创建好的 stack_list，查找出适应度次于上边的 stack，将其排为第二个 stack，后续依次累积。

对于矩形块装箱问题，也可将矩形块划分为不等宽的条带，这样就能将问题转化为条带装箱问题，只是每个条带的宽不相等，会在相应模型中多产生一个变量，同时增加一个条带宽度总和不大于板材原片宽度的边界条件，由条带构建的举行件排样示意图如图 3-7 所示。

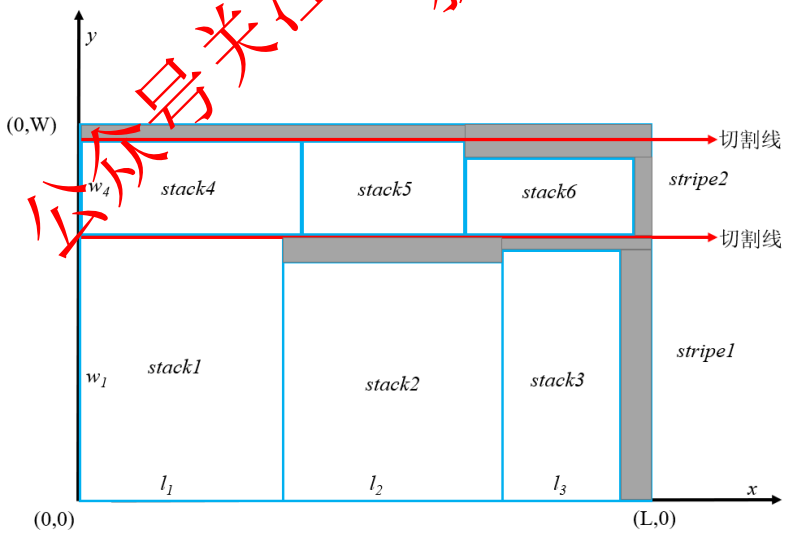

图 3- 7　总体排样图

从图中可以看出 stack1、stack2、stack3 生成了条带 stripe1，其宽由 stack1 的 w_1 决定；同时 stack4、stack5、stack6 生成了条带 stripe2，其宽由 stack4 的 w_4 决定。图中可以看出 stripe1 与 stripe2 占满了整个原板，构成一种排样方案，总体排样如图 3- 所示。且此种排样满足一刀切、三段切割、精确排样条件。

4. 问题一算法实现与求解

4.1 问题一的算法实现

在问题一的模型建立过程中，涉及到了三种算法，分别是构造二元组块算法、改进普通条带生成算法、改进背包算法（模型求解算法）。下面将逐一对这三个算法进行设计和分析，并用所给数据对算法进行实现。

4.1.1 构造二元组块算法（2-Items）

二元组块的构造算法（下文用 2-Items 表示）[3]，2-Items 算法流程框图如 4-1 所示：

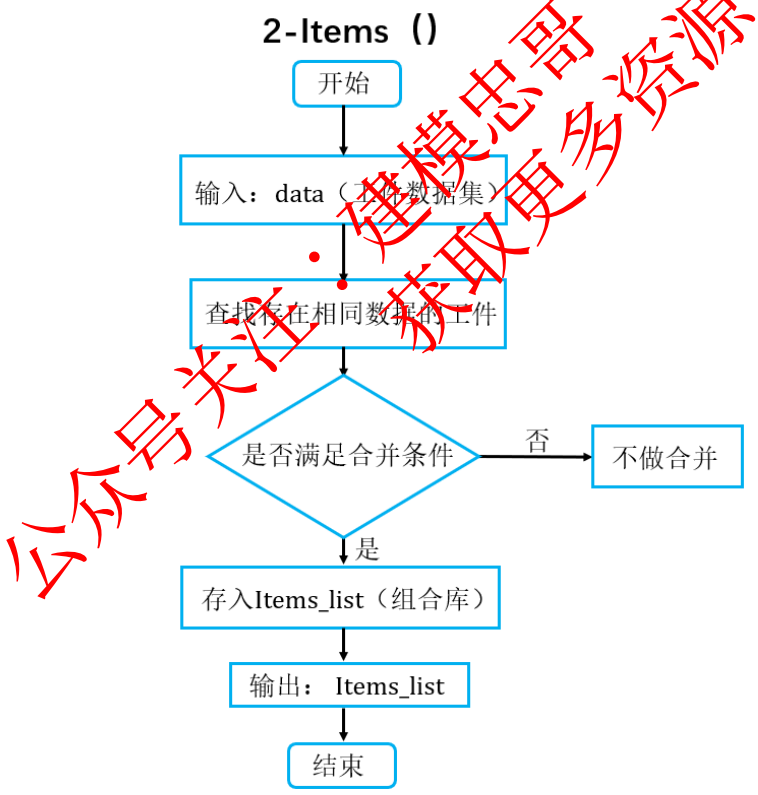

图 4- 1　2-Items 算法流程图

算法通过 Python 实现，程序代码见附录 1，在查找相同数据时，将所 item 的长和宽进行对对比，查找出有长或宽相等的 item 和其 item_id，把此类 item 进行两两组合，并判断不相等边之和是否不大于板材元件的最大边，满足条件进行合并成 2-items，并录入二元组合数据（Items_list）中，最后将 item 与 2-items 合并，得到处理后的 Items_list（数据）。

算法 1：构建二元组块算法

Input：item 数据集 dataA，板材原片数据 raw，组合类型 type，空组化数据集
Output：Items_list 数据集
1：设定最大组块所含的 item 个数为 2
2：提取 dataA 数据里面的长、宽和 id，分别搜索长相等和宽相等的 item 和其 Id
3：判断组合是否超出边界条件
4：把长相等的两个 item 进行组合得到 2-items，并标注此类合并为 1，把宽相等的两个 item 进行组合得到 2-items，并标注此类合并为 0，并存入 Items_list 中
5：查找未组合的每个 item 的长是否与其它 item 宽交叉相等的，若满足交叉条件，则进行组合并存入 Items_list 中
6：返回 Items_list 数据集

2-Items()算法是基于 python 语言编程，算法调试设备为笔记本电脑，操作系统为 Windows10，CPU 为 Core i7，算法运算时间为 2 秒，2-Items()。运算结果显示相对于原输入 dataA 数据集的 1337 个单个 item，输出结果 Items_list 数据集中罗列了所有可能组合的 item，和他们的组合方式，其中一个 item 可能和多块 item 进行组合，因此共得出有 2752 种组合可能性。因还不能确定那种，每个 item 的具体的组合方式需要在排样时进一步确定，并且略之，然后建立组合后的 Items_list 数据集。

4.1.2 启发式条带生成算法(Improved-Stripe)

普通条带模型的所有条带都是同一规格，条带都为同一宽度。本题的条带宽度根据 stack 的宽度改变，每个 stripe 的宽度由左边第一个 stack 的宽度决定，而条带的长就为原板的长（L），因此需对普通条带进行相应改进。改进点为：在模型中加入一个宽度变量，把固定宽度改为可变宽度，构造 Improved-Stripe 算法。

Improved-Stripe 启发式算法是基于最佳适应度策略（Best_fit）的组块评价机制，在排样下一块组块时，通过计算每一组块的适应度，选择适应度最高的组件进行排样，每排一次就计算一次适应度，以这种方式对每次选择的组块进行求解，确保每次排样的是最适合的。适应度直接和面积相关，通常适应度高的即为面积较大的组块，因大面积组块越靠后排样越不好找小面积组块与之搭配，这样可能导致大面积组块单独用一张原料片，不利于利用率最大化。因此设计了基于条带生成的 Improved-Stripe 启发式算法。

Improved-Stripe 启发策略：

（1） 遍历 stack_list 数据集，查找出面积最大的 stack，将其置为条带的第一个组件，按照求解第一适应度适应度的函数将 stack_list 剩下的所有 stack 进行第一适应度计算，生成第一适应度数据并降序排列，第一适应度计算见式 4-1。

$$F1_i = \frac{\text{最大面积} stack_i}{\text{原片面积}} \quad i = 1,2,3,\dots,N \tag{4-1}$$

（2） 再遍历 stack_list 数据集，在计算所有 stack 的第二适应度，生成第二适应度数据集，也对其降序排列，计算第二适应度见式 4-2，其中 k 为 stack 与原片相等边的条数（取值有 0，1，2）。

$$F2_j = k_j + \frac{stack_j\text{面积}}{\text{原片面积}} - \frac{stack_j\text{中}item\text{个数}}{1000} \tag{4-2}$$

（3） 在对条带生成过程中，优先从第一适应度数据集里选择第一适应度的最大值的 stack 首先排放，在第一适应度里数据排放完后，再从第二适应度从高到低进行排放，直到把原片再也放不下任何一个 stack，就算排样完成。

Improved-Stripe 启发式算法流程如图 4-2 所示。

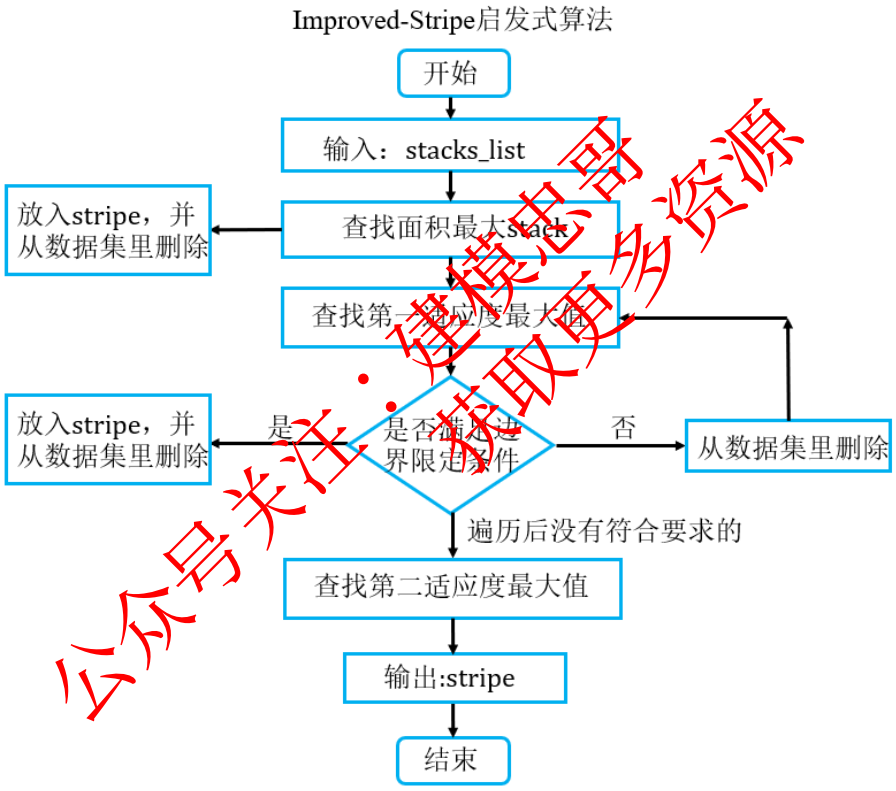

图 4- 2　Improved-Stripe 流程图

Improved-Stripe 启发式算法具体思路如图 4- 所示，首先对已经建立好的 stack 数据集进行查找，查找出面积最大的，放置在左下角，将其较长边放置在原板建立的笛卡尔坐标系的 x 轴，短边与 y 轴重合，后续查找出的第一适应度数据集中 FI_i 最大值进行排放，下面介绍 Improved-Stripe 算法具体的实现过程。

算法 2：启发式改进条带算法（Improved-Stripe 算法）

Input：stack_list 数据集
stripe 中 stack 的 id 存放数组

Output：stripe 数据集

1：设定 stripe 中 stack 长度综合小于原板材长度 L，宽度小于板材宽度 W

2：提取 stack_list 数据里面的长、宽和 id，分别求出每个 stack 的 $F1_i$

3：查找出适应度最大 stack，摆放方式为倒置（长边与 x 轴重合），并将其信息并保留到 stripe 中，stripe 中 stack 数量增加 1，并把这个 stack 从第一适应度数据集移除

4：遍历剩下的第一适应度数据集，查找出 $F1_{i+1}$，放入 stripe 中

5：判断 stripe 中的 stack 是否满足边界条件，满足边界条件就放入 stripe 中，并跳回 Step3 继续寻找下一个 stcak，不满足边界条件把这个 stack 从 stack_list 移除，也跳回 Step3，直到第一适应度数据集无符合条件的 stack，跳出循环

6：直到第一适应度数据集无符合条件的 stack，接着再以 Step5 同样的方式在第二适应度数据集里查找最大 $F2_i$，直到第二适应度数据集也无符合条件的 stack，跳出循环

7：返回 stripe 数据集

Improved-Stripe 算法是用 Matlab 编写的（代码作附录 2）法调试设备为笔记本电脑，操作系统为 Windows10，CPU 为 Core i7，算法运算时间为 2 秒，Improved-Stripe 运算得出对 dataA1 数据集生成符合要求的 stripe 共 166 个。

表 4-1 生成条带统计表

数据集	dataA1	dataA2	dataA3	dataA4
Stripe	229	201	227	222

4.2 问题一的排样和求解

4.2.1 最后的排样结果

以数据集 dataA1 为例，通过上述 Improved-Stripe 算法生成了 166 个 stripe，得到了每个 stripe 的宽度，和每个 stripe 里面包含的 item。以模型中的两个约束条件为判断依据（总宽度不能超过板材的宽度），按照宽度从大到小依次查找出每个符合条件的 stripe，并组成完整的板材排样，具体排样思路见如图 4-3 所示。

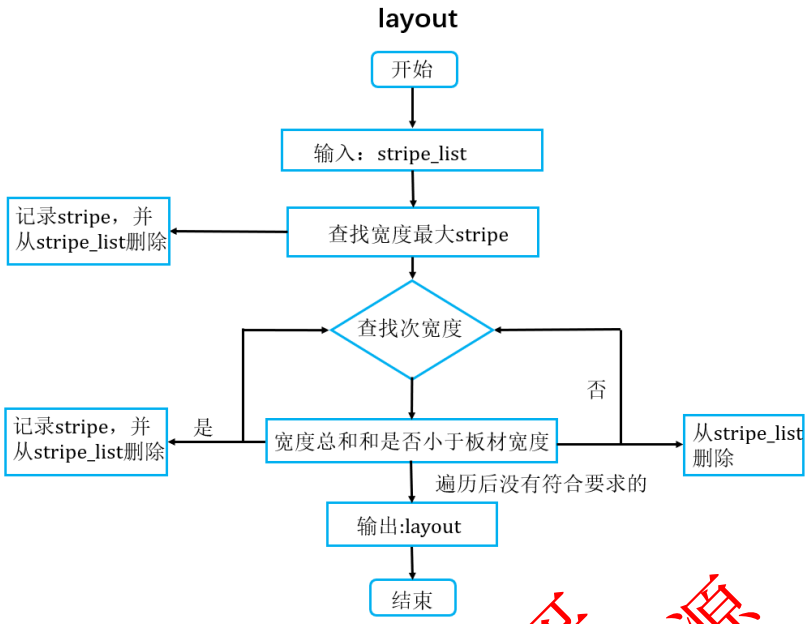

图 4- 3　排样流程图流程图

排样的算法与 Improved-Stripe 的算法类似，这里不再赘述，程序详情件附录 3 终的排样所需原片结果见表 4-2。

表 4-2 排样所需原片数统计表

数据集	dataA1	dataA2	dataA3	dataA4
原片数	89	89	88	87

最终的排样的输出结果如表 4-3 所示，我们按照产品的 id 升序进行排列，dataA 中不区分材质，第一列都一样，第二列为每个产品对应的原片序号，第三列、第四列分别为产品左下角点在原片上的坐标。通过这样的表格，就能准确给每个产品进行定位。

表 4-3 排样所需原片数统计表

原片材质	原片	产品 id	x 坐标	y 坐标	长度	宽度
YW10-0218S	14	0	2001	1216	2001	58
YW10-0219S	51	1	1339	782.5	1339	352.5
YW10-0220S	17	2	2147	517	2147	517
YW10-0221S	54	3	2377.5	1197	1179.5	702
YW10-0222S	28	4	1998	1137	1998	558
YW10-0223S	10	5	2421	568	170	568
YW10-0224S	70	6	2315	358	769	358
YW10-0225S	85	7	805	1211.5	343	92.5
YW10-0226S	4	8	2348	630	2348	630

为了能更直观的看出排样结果是否满足条件，下面将在排样后的数据集里挑选两两块原片的排样结果图，选择了 dataA1 排样结果里的第 1 块与安排的和第 88 块原片的数据。第一块原片的排样数据如表 4-4 所示。

表 4-4 datA1 第 1 块原片的排样数据表

原片材质	原片	产品 id	x 坐标	y 坐标	长度	宽度
YW10-0243S	1	25	2398	511	2398	299
YW10-0279S	1	61	2367	1193	2367	218
YW10-0291S	1	73	2388	975	2388	98
YW10-0328S	1	113	2398	839	2398	328
YW10-0384S	1	170	2393	877	2393	38
YW10-0448S	1	235	2418	58	2418	58
YW10-0564S	1	356	2418	116	2418	58
YW10-0744S	1	542	2418	174	2418	58
YW10-0920S	1	723	2416	212	2416	38

表 4-3 中，总计排布了 9 个 item，并记录了每个 item 的右上角点坐标，只需要找到每个点，然后用长和宽画出每个 item。采用 AutoCAD 画出了第 1 块原片的效果图，如图 4-4 所示。

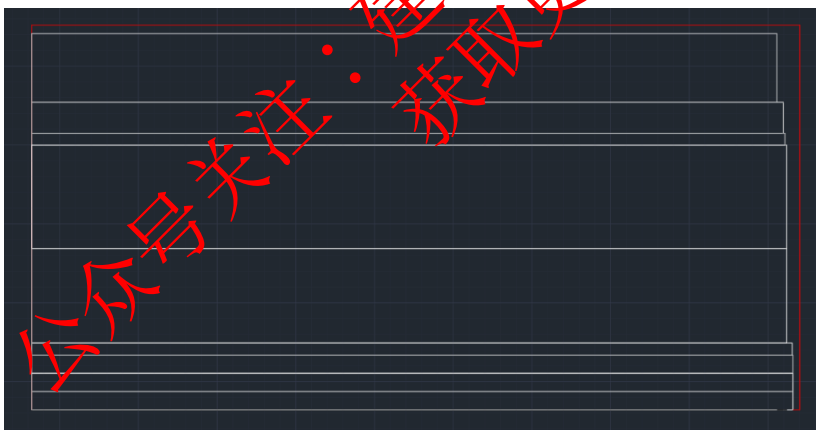

图 4- 4datA1 第一块原片的排样图

图中白色矩形代表一个 item，红色线条为原片边界，可以看到图中从下往上数共有 9 个白色矩形，和表中的个数相符合，这块原片排样都选择的是大块 item，因此只有一个 stack，满足三阶段、一刀切、精确排样的条件。

dataA1 数据中第 88 块原片排样结果数据如表 4-5 所示。

表 4-5 datA1 第 88 块原片的排样数据表

原片材质	原片	产品 id	x 坐标	y 坐标	长度	宽度
YW10-0285S	88	67	2164	1212	338	158
YW10-0351S	88	136	2432	439	245	439
YW10-0360S	88	145	2428	855	241	416
YW10-0370S	88	156	1104	327	361	327
YW10-0403S	88	189	370	1036	370	308

YW10-0435S	88	222	1104	835	361	508
YW10-0469S	88	256	1465	508	361	508
YW10-0522S	88	311	1465	1016	361	508
YW10-0547S	88	337	743	439	364	439
YW10-0560S	88	351	1826	830	361	830
YW10-0567S	88	359	2187	527	361	527
YW10-0588S	88	380	363	1208.5	363	172.5
YW10-0630S	88	423	379	728	379	728
YW10-0641S	88	434	1457	1148.5	353	132.5
YW10-0808S	88	607	743	778	364	339
YW10-0853S	88	653	2187	1054	361	527
YW10-0903S	88	703	1825.5	1177	360.5	347
YW10-0907S	88	710	1104	1162	361	327
YW10-0913S	88	716	743	1117	364	339

表 4-5 中，总计排布了 19 个 item，并记录了每个 item 的右上角点坐标，只需要找到每个点，然后用长和宽画出每个 item。采用 AutoCAD 画出了第 88 块原片的效果图，如图 4-5 所示。

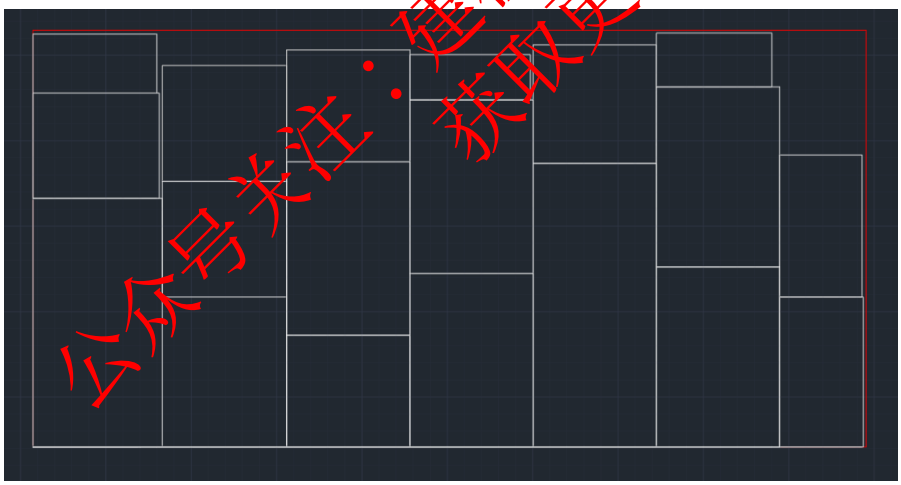

图 4- 5datA1 第 88 块原片的排样图

图中白色矩形代表一个 item，红色线条为原片边界，可以看到图中共有 19 个白色矩形紧密排布，和表中的 item 个数相符合，且所有 item 都紧密排在左下角，且满足三阶段、一刀切、精确排样的条件。

4.2.2 求解板材利用率

这里暂以 dataA1 数据集为例（其他几组结果见后续表格），用所建立的改进利用率求解模型求出每片板材的利用率。总的利用率和每片板材的利用率之间的关系由下面公式给出：

$$E_{total} = \sum_j^J E_j \qquad j = 1,2,3 \ldots, J \tag{4-1}$$

$$E_j = \frac{\sum_i^I l_i w_i}{L \times W} \quad i = 1,2,3 \ldots, i \tag{4-2}$$

其中 E_{tatal} 为排样完 dataA1 所有 item 的总利用率，j 代表第 j 块板子，l_i 表示第 i 块 item 的长度，w_i 表示第 i 块 item 的宽度，L 表示原板材的长度，W 表示原板材的长度。

通过 Matlab 程序（程序见附录 4 利用率进行求解，求解结果见表 4-6。

表 4-6 利用率统计表

数据集	dataA1	dataA2	dataA3	dataA4
原片数	89	89	88	87
利用率	93.87%	93.12%	95.15%	94.08%

5. 问题一算法的评估和对比

5.1 问题一的模型评估

本文的模型是基于求解最大价值的背包模型的改进模型，在传统背包模型中每个物件都有价值，本题方形件无参评价值，所有价值均为 1，再借鉴背包模型的目标函数建立本文的求最大利用率的目标函数。通过借鉴加改进的方式建立的目标模型具有可行性，而且模型的算法难度不大。

最大利用率模型建立后，把问题难度转向如何排样使得原片的利用率最大，我们采取的策略是，在满足二刀切、三段式、精确切割条件下合并满足合并条件的最小单元，把两个 item 合并为一个 2-items，把 2-items 合并为一个 stake,再把多个 stake 合并为一个 stripe，采用这样由小到大的合并方式进行排样，这样的排样方式刚好为切割方式的一个逆向过程，在逻辑上和可行性上都是行得通的，因此可以说本文所建立的排样模型是能够解决问题的，而且所得到的利用率都在 90%以上，可见排样效率高，同时也论证了排样方案的可行性。

总本文建立的模型有：两个用于合并组件的模块、一个用于排样的模块、一个用于计算排样后的利用率模块，三个模块的复杂度都不高，而且求解难度也较小。

5.2 问题一的算法评估

5.2.1 算法的可行性分析

本文所使用的算法（2-items，Improved-Stripe）为自行构建的一种实用性算法。现目前针对矩形件排样的主要算法有基于 Best-Fit 策略的适应度择优匹配法、双队列推荐适应度择优匹配法、组块双递归排样法、贪心搜索法。本文所用的算法思想与 Best-Fit 策略的适应度择优匹配法的思想相似，都是以方形件的面积或长短为依据寻找下一块合适的方形件进行排样，从这个角度看，本文所建立的方法是可行的。

5.2.2 算法的复杂度分析

本题中合成算法 2-items、Improved-Stripe、排样用的算法三种都用的是快速生

成算法[4]，每个算法都需遍历所有最小单元（即所有最小单元的长度和宽度），因此时间复杂度为 $O(nLW)$，其中 n 为最小单元的种数，L 为数据集的长度数据总量，W 为宽度数据总量，排样并计算完四组所用总时间为 3.9748 秒。

6. 问题二的分析

6.1 问题二分析

问题二要求建立混合整数规划模型解决订单组批问题并求解最大利用率，题目要求满足问题一的约束条件，意味着可以沿用第一题的排样模型，只需要将第二问的数据进行提前分批次，然后利用第一题的排样模型进行排样并计算每个数据集需要的原片个数和原片利用率。先分析所给数据，以数据 dataB1 为例，数据总计 26812 个 item，其中有 100 多种材料，几百个订单，数据量较大。按照题目要求每个批次最多 1000 个 item，理想情况下需要 27 批次。再分析约束条件，每个订单必须在一个批次完成，每个批次必须满足每个批次切割产品项（item）总数不超过 1000 个，每个批次切割的产品项总面积不超过 250m^2。观察数据集中的订单多分布在百种左右，如果一个批次仅完成一个订单的话，那么会使得工厂效率非常低下，板材利用率也不高。所以必须考虑在满足约束条件下采用何种方式将订单分配到不同的批次。综上所述既要满足约束条件，又要合理地将订单分到不同批次，同时尽可能使得原片利用率较高是解决问题的关键。问题二求解算法流程图 6-1。

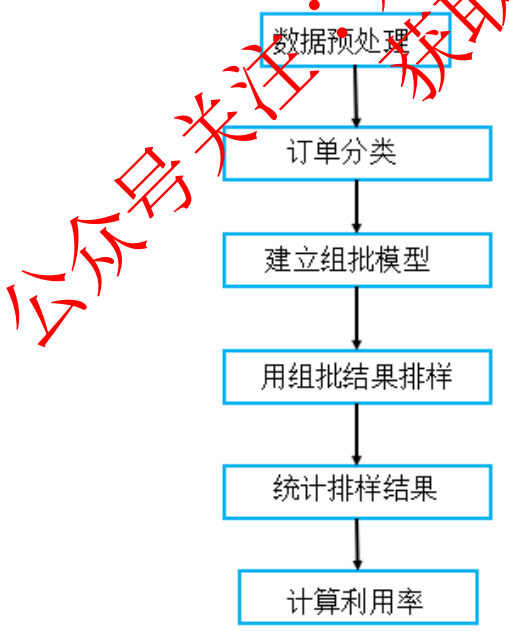

图 6-1 问题二思路流程图

6.2 问题二模型建立

6.2.1 对数据进行订单分类

观察数据，发现数据中的订单号是散乱的，结合题目约束条件（每个批次必

须完整包含一个订单），应该把不同产品项按照订单（order）进行归类，按照订单号把同一订单号的产品项归为一类。例如数据集 dataB1 其中有多个产品项多个订单。对 5 个数据集分别进行订单归类，归类结果见表 6-1。

表 6-1 订单归类统计表

数据集	dataB1	dataB2	dataB3	dataB4	dataB5
订单个数	546	381	410	381	604

6.2.2 问题二组批优化模型建立

问题二最终问题归结于最优组批使得板材利用率最大化，同样以利用率为目标函数建立混合整数规划模型：

$$E = \frac{\sum_i^n S_i}{K \times L \times W} \tag{6-1}$$

其中E表示利用率，i 表示第 i 个订单，S_i表示第i个订单所包含产品项的总面积，K 表示该数据集需要的原片总个数，L 表示原片长度，W 表示原片宽度。

模型约束条件为：

（1）订单的分批结果，若订单X_i分到第j个批次，则记$x_{ij} = 1$,否则为 0

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1,2,3,\dots,n \tag{6-2}$$

（2）每个批次 item 的总量不能超过 1000 个

$$\sum_{x_i \in T_j} n_i \le 1000 \quad j = 1,2,3,\dots,T \tag{6-3}$$

（3）每个批次所包含的 item 总面积不能超过 250m^2

$$\sum_{x_i \in T_j} S_i \le 2.5 \times 10^8 \quad j = 1,2,3,\dots,T \tag{6-4}$$

其中X_i表示第 i 个订单；T 为批次（$T \leqslant n$，n 为该数据集中的订单总数），T_j 表示第 j 个批次；x_{ij}表示订单X_i的分批结果，x_{ij}的取值只有 1 和 0，1 代表订单X_i被分到第j个批次，0 代表订单X_i没有被分到第j个批次；n_i表示订单X_i中 item 的个数；S_i表示订单X_i所包含 item 的总面积。

以利用率为目标函数建立混合整数规划模型为：

$$\max E = \frac{\sum_i^n S_i}{K \times L \times W}$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{n} x_{ij} = 1 \quad i = 1,2,3,\dots,n \\ x_{ij} = \begin{cases} 1 & x_i \in T_j \\ 0 & x_i \notin T_j \end{cases} \\ \sum_{x_i \in T_j} n_i \le 1000 \quad j = 1,2,3,\dots,T \\ \sum_{x_i \in T_j} S_i \le 2.5 \times 10^8 \quad j = 1,2,3,\dots,T \end{cases} \tag{6-5}$$

6.2.3 基于变异系数法的求解模型

涉及到订单分批问题，自然考虑聚类思想，将有相同属性的不同订单聚为同一类，之后再根据聚类结果递归合并订单，直至产生所有的批次。

我们以相似度作为聚类的依据。相似度用于衡量两个订单之间的联系紧密程度，例如是否有相同材料的产品项以及相同材料的产品项数量。相似度大在一定程度上意味着合并在一个批次里将会更节省原片消耗，提高原片利用率。

通过经验分析，我们认为两个订单间相似度主要受以下三个指标的影响。三个指标如下：

1）两个订单拥有相同材料的种类数，记作M^1

2）两个订单拥有相同材料的产品项个数，记作M^2

3）两个订单拥有不同材料的种类数，记作M^3

我们采用变异系数法对上述三个指标赋权重[5]，从而获得相似度的计算公式。

变异系数法是一种利用指标的变异程度来确定指标权重的方法，可以实现对被评价对象的每个指标动态赋予权值。如果评价对象的变异程度越大，说明其影响能力越强，越重要，应该加大其权重；如果一个指标变异程度小，应赋予小的权重。

变异系数法涉及到两个概念：正向指标与负向指标，正向指标值的是指标越大评价结果越好，负向指标指的是指标越大评价结果越差。

负向指标正向化计算公式为：

$$Neg_{ij} = \frac{1}{k+max|x_j|+x_{ij}} \tag{6-6}$$

在 I 个评价对象和 J 个评价指标形成的 $(x_{ij})_{I\times J}$ 中，Neg_{ij} 即为负向指标，其余的为正向指标。上述三个指标中中M^1，M^2为正向指标，M^3为负向指标。

因为M^1、M^2、M^3这三个指标的数量级不一样，需要对三个指标进行标准化。标准化公见式 6-7。

$$r_{ij} = \frac{Neg'_{ij}}{\sqrt{\sum_{i=1}^{I} Neg'^{\,2}_{ij}}} \tag{6-7}$$

并得到标准化后的矩阵 $R = (r_{ij})_{I\times J}$。

指标均值计算公式：

$$A_j = \frac{1}{n}\sum_{i=1}^{I} r_{ij} \tag{6-8}$$

指标标准差计算公式：

$$S_j = \frac{1}{n}\sum_{i=1}^{I} r_{ij}\sqrt{\frac{1}{n}\sum_{i=1}^{I}(r_{ij}-A_j)^2} \tag{6-9}$$

求变异系数公式：

$$V_j = \frac{S_j}{A_j} \tag{6-10}$$

权重公式：

$$\omega_{ij} = \frac{V_j}{\sum_{j=1}^{J} V_j} \tag{6-11}$$

通过编程实现，我们利用变异系数法找到了 3 个指标的最佳权重，分别为 0.4780，0.4955，0.0265。由于前两个指标权重很大，占据了 97.35%，同时公式里

负向指标正向化误差较大，所以这里我们舍弃了第三个指标，仅采用前两个指标来计算相似度。

不妨设订单 i 的产品项总数为 X_i，订单 j 品项总数为 X_j，两个订单中具有相同材料属性的 item 总数为 Y ,两个订单共需要材料 k 种，相同材料 k_{ij} 种。

相似度自定义计算公式如下：

$$Similarity = 0.4780 * \frac{k_{ij}}{k} + 0.4955 * \frac{Y}{X_i+X_i} \tag{6-2}$$

6.3 问题二算法实现与求解

6.3.1 基于相似度聚类的订单分批次排样算法

上文我们建立了相似度模型，得到了相似度的计算公式。接着将相似度作为聚类标准，在满足约束条件的基础上，合并相似度较高的两个订单,不断生成更大的新订单，在递归的思路基础上，把订单合并到不同批次里，直至生成全部批次。基于相似度聚类的订单分批次算法如下：

算法 3：基于相似度聚类的订单分批次算法

Input：所有订单（order）和订单信息，包含该订单的产品项及对应的材料

Output：所有订单的组批结果

1：数据集处理：统计每个订单的信息，如产品项（item）总数、材料数等

2：选择初始化订单

3：计算该订单与剩余订单的 Similarity。

4：从剩余订单中选择与当前订单相似度最高的订单。若合在一起满足约束条件则合并；若不满足选择相似度次高的订单判断是否可以合并，直至合并成一个新订单。

5：对于合并出来的新订单，重复 step3，step4，直至生成一个批次。

6：返回 step2，生成下一个批次，直至所有订单都被分批。

订单分好批次后，将每个批次的结果汇总，并按照材料不同进行分类，对于同一批次同种材料的所有产品项采用问题一的排样算法，生成比较优秀的切割方案和排样版式。

自此，基于相似度聚类的订单分批次排样算法就全部结束了。用 Python 对算法进行了实现，具体程序代码见附录 5，完整代码与输出结果见附件。以下是整个过程的流程图：

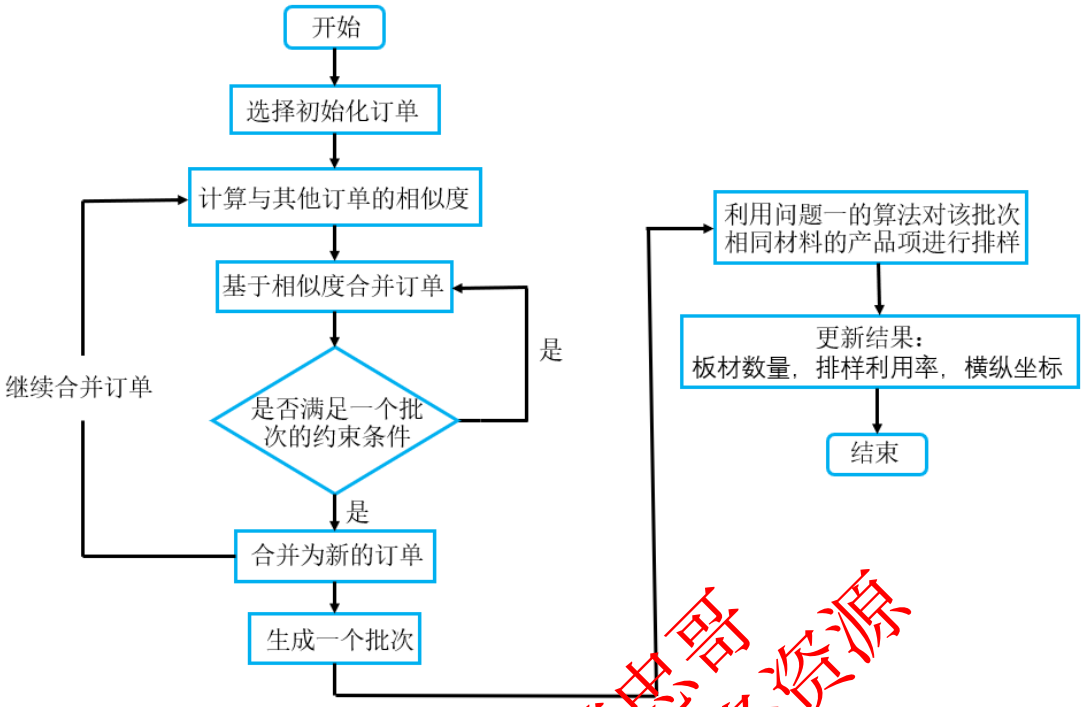

图 6-2 订单分批次流程图

6.3.2 问题二的分批、排样结果

用 Matlab 对算法进行了实现，具体程序代码见附录 6，完整代码与输出结果见附件。以数据集 dataB1 为例，该数据集共有 26811 个产品项，546 个订单，48 种材料数。经过我们的组批算法得到 42 个批次，通过排样计算，耗损 3462 块原片。原片利用率为 86.03%。

表 6-2 dataB1 排样结果样表

批次	材质	原片序号	产品 id	x 坐标	y 坐标	长度	宽度
28	YW1-0218S	2118	0	2220	761.5	858	337.5
36	YW1-0218S	1345	1	425.5	1214	425.5	192
36	YW1-0218S	1332	2	390	1164	195	48
9	YW1-0215S	2861	3	2409.5	933	302	480
36	YW1-0215S	1330	4	1022	1182	1022	330
9	YW1-0218S	2834	5	781	1148.5	781	58
9	YW1-0218S	2838	6	1483	972	1483	269
9	YMH-0215S	2879	7	1076.5	1047.5	405	92.5
36	5-0218S	1366	8	2392	466.5	781	466.5

以 dataB1 数据集为例，我们展示了组批排样结束后第 0 批次的相关信息，该批次共包含 9 个订单，5 种材料，共消耗 93 块原片，具体信息见表 6-3。

表 6-3 dataB1 数据集第 0 个批次数据展示

订单数	具体订单	材料数	具体材料	该种材料消耗原片数
9	order443	5	QKQ	48
	order 455			
	Order458		NEQl	1
	Order490			
	Order491		JH	1
	Order493			
	Order517		FMB	33
	Order518			
	Order519		PKQL	10

最后分别对 dataB1、dataB2、dataB3、dataB4、dataB5 进行组批、排样、利用率计算，得出 5 个数据的利用率都在 80%以上，最终结果汇总见表 6-4。

表 6-4 第二题结果统计表

数据集	dataB1	dataB2	dataB3	dataB4	dataB5
批次总数	42	12	27	30	45
原片总数	3462	2270	2298	2392	3733
板材利用率	86.03%	84.88%	84.15%	84.87%	82.77%

7. 问题二算法的评估和对比

7.1 问题二的模型评估

第二题沿用第一题的排样方案，因此在第二问不在考虑排样问题，因此把注意力放在建立组批模型。建立的组批模型由两部分组成，第一个模型是组批优化求解最大利用率模型，也是求解本题的目标模型，此模型的优劣依赖于组批结果，组批合理利用率自然就大。第二个模型是基于变异系数法的相似度求解模型，此模型依赖于所求三个指标权重的准确性，指标权重的准确性高，相似度也就越准确，组批的结果越好。

依照相似度进行组批是解决组批问题的一种常用方法，模型可行性高，基于背包算法的最大最大利用率求解模型也是常用方法，且在第一文中已验证其对本题具有很高的适用性，这说明第二问建立的模型是可靠的。

7.2 问题二的算法评估

7.2.1 算法的可行性分析

第二问是在第一问的基础上增加了基于变异系数法的聚类组批算法，此算法的根据是相似度，通过求解评价指标的权重，通过权重计算出订单之间的相似度，进行组批。算法的核心是计算评价指标的权重，求解变异系数法的权重的难点在于权重是根据产品数据进行变动的。通过建立的求解模型，可根据数据集首先找出负向指标，根据负向指标求出均值，再逐一求出标准差、变异系数，最后求出权值，算

法求解过程条理清晰且严谨，是可行的。

7.2.2 算法的复杂度分析

本题中变基于异系数法的聚类算法也是快速生成算法，每个算法都需遍历所有最小单元（进行负向指标计算），因此时间复杂度为 $O(nLWK)$，其中 n 为指标的量数，L 为数据集的长度数据总量，W 为宽度数据总量，K 为排样批次，排样并计算完 5 组所用总时间为 156 秒。

8. 总结

8.1 模型和算法的优点

1.建立的排样模型是依据由小到大进行合并的理念进行的，是一个三阶段切割的逆过程，与实际切割情况贴切。

2.所建立算法为实用性算法，算法具有较高的移植性和复现性。

3.对于构造的启发式算法，是基于最佳适应度的推优算法，优先选出最佳组件，排样结果利用率都在 95%左右。

4.模型对应的算法都为快速生成算法，相比于元启发式算法，我们构造的算法计算复杂度更小，计算时间短。

8.2 模型和算法的缺点

1.问题二建立的组批模型的聚类标准为自定义的相似度，相似度的计算公式大大影响组批效果。若有更好的相似度计算公式，问题二的结果将会有进一步的改善。

2.在算法实现的过程中，选择好的数据结构来存储数据将会大大影响代码效率。例如用列表（python），元胞数组和矩阵（matlab）来存储数据。若有更好选择，在处理大规模数据集时将会大大提升计算效率。

8.3 总结与推广

通过查阅相关文献，了解到组批优化和排样优化问题是运筹学中的经典问题，同时也是 NP 难题。本文抓准两个问题的要求和约束条件，针对问题一建立混合整数规划模型，提出构建二元组块算法和启发式条带生成算法解决了排样优化的问题，满足“一刀切、三阶段、精准切割”的条件；针对问题二建立混合整数规划模型和基于变异系数法的相似度模型，提出基于相似度聚类的订单分批次算法解决组批优化的问题，每个批次都满足“订单不分割，材料分开排，产品项有约束”的条件。本文也取得不错的结果，问题一的 4 个数据集板材利用率分别为 93.87%，93.12%，95.15%，94.08%，问题二的 5 个数据集板材利用率分别为 86.03%，84.88%，84.15%，84.87%，82.77%。

目前在组批优化和排样优化问题上已有一些成熟的模型和算法，同时也有一些商业软件专门提供组批规划和排样规划服务。如果能有更多时间调研研究，集众家之所优，算法效果会更加好。

单论本文算法，针对排样问题，可以尝试建立其他启发式排样算法，比较哪种排样策略性能更好；针对组批问题，可以在相似度公式上进行进一步分析，得到更加科学合理的计算公式。同时，在程序设计方面，一方面，选择合适的数据结构存储数据可以使得代码效率更高；另一方面，养成良好的编程习惯，注意细节也可以提高代码效率，提升可读性（例如在 matlab 中预先开辟空间计算效率远高于不预设空间）。以上这些方面是我们思考可以改进和完善的地方。

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.
[2] 李立平. 二维三阶段排样算法研究[D]. 广西大学，2016.
[3] 王磊. 板构产品制造过程中的智能排样优化方法研究[D]. 广东工业大学，2017.
[4] 算法复杂度的评估以及常用函数的复杂度计算 www.t.zoukankan.com/xiaoyh-p-10259283.html，2017.
[5] 张文朝,顾雪平.应用变异系数法和逼近理想解排序法的风电场综合评价[J].电网技术,2014,38(10):2741-2746.
[6] 曹炬,周济.矩形件排样优化的一种近似算法[J．计算机辅助设计与图形学学报,1995,7(3):190-195.
[7] Berkey J o, Wang P Y. Two-dimensional finite bin-packing algorithms[J]. Journal of theOperational Research Society,1987,38(5):423-429.
[8] Chan T M, Alvelos F, Silva E, Valerio de Carvalho J M.Heuristics with stochastic neighborhoodstructures for two-dimensional bin packing and cutting stock problems[J]. Journal of OperationalResearch,2011, 28(2):255--278.

附录

附录 1 组块程序代码

```
import pandas as pd
import os
import numpy as np

L = 2440
W = 1220
S = L * W

# 题目一预处理
def predata(file):
    """
        数据格式为
         ['Id', 'Length', 'Width', 'grouptype', 'fitness', 'rotation', 'Id2']
        返回值为 list
    """
    data_file = os.path.join('E:\\19 届华为杯\\2022 年 B 题\\题目\\子问题 1 数据集 A', file)
    data = pd.read_csv(data_file)
    item_id = data.iloc[:, 0]
    item_size = data.iloc[:, 3:5]
    item_size['GroupType'] = -1
    item_size['fitness'] = 0
    item_size['rotation'] = 0
    item_size['Id2'] = -1
    item = pd.concat([item_id, item_size], axis=1).values.tolist() # 将 csv 转成列表

    # 获得组化粒度为 1 的 group
    item_group1_rt = [] # 旋转后的 grouplist
    for i in item:
        if i[1] != i[2] and i[2] <= L and i[1] <= W:
            # 长不等宽，宽小于板材长且长小于板材宽时，旋转
            ap = [i[0], i[2], i[1], -1, 0, 1, -1]
            item_group1_rt.append(ap)
    for i in item: # 去除旋转前长大于板材长或宽大于板材宽的元素
        if i[1] > L or i[2] > W:
            del i
    item_group1 = item
    for i in item_group1_rt:
        item_group1.append(i)

```

```
    # 去除组化粒度为 1 的 group 的相同元素
    item_group1_cs = item_group1.copy()
    l = len(item_group1)
    for i in range(l - 1, -1, -1):
        for j in range(i - 1, -1, -1):
            if item_group1[i][1] == item_group1[j][1] and /
                                item_group1[i][2] == item_group1[j][2]:
                del item_group1_cs[i]

    # 获得组化粒度为 2 的 group
    item_len = len(item_group1_cs)
    item_group2 = []
    for i in range(0, item_len):
        for j in range(i + 1, item_len):
            if item_group1_cs[i][1] == item_group1_cs[j][1]:
                # 如果两工件长相等，拼接并判断宽小于板材宽
                w = item_group1_cs[i][2] + item_group1_cs[j][2]
                if w < W:
                    # ['Id', 'Length', 'Width', 'grouptype', 'fitness', 'rotation', 'X', 'Y']
                    item_group2.append([item_group1_cs[i][0], item_group1_cs[i][1], w, 1, 0, item_group1_cs[i][5], item_group1_cs[
j][0]])
            if item_group1_cs[i][2] == item_group1_cs[j][2]:
                # 如果两工件宽相等，拼接并判断长小于板材长
                l = item_group1_cs[i][1] + item_group1_cs[j][1]
                if l < L:
                    item_group2.append([item_group1_cs[i][0], l, item_group1_cs[i][2], 1, 0, item_group1_cs[i][5], item_group1_cs[j
][0]])
    for i in item_group2: # 拼接两列表
        item_group1_cs.append(i)
    item_group = item_group1_cs.copy()
    return item_group

def cmpFitness1(data): # 计算 BigItem 适应度
    for i in range(0, len(data)):
        if data[i][3] != -1:
            g = data[0:i]
            break
    bigitem1 = []
        for i in g:
            if i[1] > L / 2 and i[2] > W / 2:
                bigitem_tmp = i.copy()
                bigitem_tmp[4] = i[1] * i[2] / S
                bigitem1.append(bigitem_tmp)
    bigitem1 = sorted(bigitem1, key=lambda x:x[4], reverse=True) # 按 fitness 倒序排序
```

```
    return bigitem1

def cmpFitness2(group): # 计算 NoBigItem 适应度
    nobigitem = []
    for i in group:
        if i[1] > L / 2 and i[2] > W / 2:
            continue
        nobigitem_tmp = i.copy()
        same_num = 0
        if nobigitem_tmp[1] == L:
            same_num += 1
        if nobigitem_tmp[2] == W:
            same_num += 1
        if i[3] == -1:
            sl_num = 1
        else:
            sl_num = 2
        nobigitem_tmp[4] = same_num + i[1] * i[2] / S - sl_num / 1000
        nobigitem.append(nobigitem_tmp)
        nobigitem = sorted(nobigitem, key=lambda x:x[4], reverse=True) # 按 fitness 倒序排序
    return nobigitem

data = predata('dataA1.csv')
bigitem = cmpFitness1(data)
nobigitem = cmpFitness2(data)

name = ['Id', 'Length', 'Width', 'grouptype', 'fitness', 'rotation', 'Id2']
group = pd.DataFrame(columns = name, data = data)
group.to_csv('E:\\19 届华为杯\\2022 年 B 题\\题目\\子问题 1-数据集 A\\group-A1.csv')

name = ['Id', 'Length', 'Width', 'grouptype', 'fitness', 'rotation', 'Id2']
group = pd.DataFrame(columns = name, data = bigitem + nobigitem)
group.to_csv('E:\\19 届华为杯\\2022 年 B 题\\题目\\子问题 1-数据集 A\\item-A1.csv')
```

附录 2 生成条程序代码

```
%% 生成条带
tic;
sum=0;
stripe=[];
ID={};
IDX={};
IDY={};
while(~isempty(all_data))
```

```
    sum=sum+1;
    %%假设第一段为竖直切割
    %%用 stack 去逼近条带，满足多个 stack 宽度类似，长度之和接近 2240，形成多个长条带
    data1=all_data;
    id=[];
    idx=[];
    idy=[];
    %% 排放规则
    %先放入长最大的,则这个条带的长就确定了为 L,id 记录了放入的 item 的编号
        L=data1(1,2);
        all_width=data1(1,3);
        id=[id,all_data(1,1)];
        idx=[idx,L];
        idy=[idy,all_width];
        all_data(1,:)=[];
    %% 先找长与 L 相等的，判断是否能放入
        tmp_index=find(all_data(:,2)==L);
        if ~isempty(tmp_index)
          data2=all_data;
          tmp1_index=[];
          for i=1:size(tmp_index,1)
          %判断找到的 tmp_index（i）是否满足条件
          if all_width+data2(tmp_index(i),3)<=1220
             %能放入则放入
             all_width=all_width+data2(tmp_index(i),3);
             id=[id,data2(tmp_index(i),1)];
             idx=[idx,L];
             idy=[idy,all_width];
             tmp1_index=[tmp1_index,tmp_index(i)];
          end
         end
         all_data(tmp1_index,:)=[];
       end
       %% 再找宽与 L 相等的
       tmp_index=find(all_data(:,3)==L);
       if ~isempty(tmp_index)
         data2=all_data;
         tmp1_index=[];
          for i=1:size(tmp_index,1)
          %判断找到的 tmp_index（i）是否满足条件
          if all_width+data2(tmp_index(i),2)<=1220
             %能放入则放入
             all_width=all_width+data2(tmp_index(i),2);
             id=[id,data2(tmp_index(i),1)];
```

```
            idx=[idx,L];
            idy=[idy,all_width];
            tmp1_index=[tmp1_index,tmp_index(i)];
          end
        end
        all_data(tmp1_index,:)=[];
      end
      %% 下一个长度最长的
while(all_width<1220)
    %没有与之相等的，就找下一个长度最长的
     m=1220-all_width;
    [ ~,next_ind]=max(all_data(:,3)<m);
    if isempty(next_ind)
       break;
    else
       l=all_data(next_ind(1),2);w=all_data(next_ind(1),3);
       all_width=all_width+all_data(next_ind(1),3);
    end
    if all_width>1220
       break;
    else
       id=[id,all_data(next_ind(1),1)];
       idx=[idx,l];
       idy=[idy,all_width];
       all_data(next_ind(1),:)=[];
    end
    %找长度或者宽度和这个宽度 w 相等的组件放入空隙
       all_length=l;
       while(all_length<L)
          all_length_tmp=all_length;
          %% 宽度等于 w 的组件，则放入
           tmp_index=find(all_data(:,3)==w);
            if ~isempty(tmp_index)
            data2=all_data;
            tmp1_index=[];
             for i=1:size(tmp_index,1)
              %判断找到的 tmp_index（i）是否已经用过了
              if all_length+data2(tmp_index(i),2)<=L
                 %能放入则放入
                 all_length=all_length+data2(tmp_index(i),2);
                 id=[id,data2(tmp_index(i),1)];
                 idx=[idx,all_length];
                 idy=[idy,all_width];
                 tmp1_index=[tmp1_index,tmp_index(i)];
```

```
                end
                end
                all_data(tmp1_index,:)=[];
            end
            %% 长度等于 w 的组件,则旋转放入，all_length 加的是其第三列的值 l
            tmp_index2=find(all_data(:,2)==w);
            if ~isempty(tmp_index2)
            data2=all_data;
            tmp1_index2=[];
             for i=1:size(tmp_index2,1)
             %判断找到的 tmp_index（i）是否已经用过了
             if all_length+data2(tmp_index2(i),3)<=L
                %能放入则放入
                all_length=all_length+data2(tmp_index2(i),3);
                id=[id,-1*data2(tmp_index2(i),1)];
                idx=[idx,all_length];
                idy=[idy,all_width];
                tmp1_index2=[tmp1_index2,tmp_index2(i)];
             end
             end
             all_data(tmp1_index2,:)=[];
            end
            [~,other,~]=intersect(all_data(:,1),abs(id));
            all_data(other,:)=[];
            if all_length==all_length_tmp %说明此次循环未放入任何组件，则跳出循环
              break;
            end
        end
    end
    stripe=[stripe,L];
    ID=[ID,{id}];
    IDX=[IDX,{idx}];
    IDY=[IDY,{idy}];
end
time2=toc;
```

附录 3 排样程序代码

```
%% 生成原片
tic;
n=size(stripe,2);
sum_ban=0;
num_id=0;
IID={};
```

```
IIDX={};
IIDY={};
while(num_id<n)
    sum_ban=sum_ban+1;
    id=[];
    length=0;
    tmp_ID=[];
    tmp_IDX=[];
    tmp_IDY=[];
    while(length<2440)
        index=find(stripe(1,:)<=(2440-length));
        tmp=find(stripe(index)==-1);
        index(tmp)=[];
        if ~isempty(index)
            length=length+stripe(1,index(1));
            if length<=2440
                id=[id,index(1)];
                tmp_ID=[tmp_ID,cell2mat(ID(index(1)))];
                tmp_IDX=[tmp_IDX,cell2mat(IDX(index(1)))+length-stripe(1,index(1))];
                tmp_IDY=[tmp_IDY,cell2mat(IDY(index(1)))];
                stripe(1,index(1))=-1;
            end
        else
            break;
        end
    end
    IID=[IID,{tmp_ID}];
    IIDX=[IIDX,{tmp_IDX}];
    IIDY=[IIDY,{tmp_IDY}];
    num_id=num_id+size(id,2);
end
time3=toc;
```

附录 4 利用率计算程序代码

```
a=dataA(:,2).*dataA(:,3);
p=sum_ban;
ratio=sum(a)/(p*2440*1220)
```

附录 5 组批程序代码

```
import pandas as pd
import numpy as np

```

```
data_origin = pd.read_csv('E:\\19 届华为杯\\2022 年 B 题\\题目\\子问题 2-数据集 B\\dataB1.csv')
material = data_origin.iloc[:, 1].values.tolist()  # 读取 material 列
material = list(set(material))
material_dict = {}
num = 1
for i in material:  # 制作 material 字典
    material_dict[i] = num
    num += 1

del data_origin['item_num']
data = data_origin.values  # 删除 item_num 列

for i in data: # 将材料替换成数字
    i[1] = material_dict[i[1]]

data = data[np.argsort(data[:,4])]
order_list = []
for i in range(1, len(set(data[:,4])) + 1):
    index = np.array(np.where(data[:,4] == i))
    order_list.append([[i], data[index[0], 0:4]]) # [[order], [[id, material, l, w]]]

def similarity(group1, group2):
    # 查找相同材料
    num = 0
    num2 = 0
    for i in range(1, 131):
        if group1[1][:, 1].__contains__(i) and group2[1][:, 1].__contains__(i):
            num += (len(np.array(np.where(group1[1][:, 1] == i))[0]) + len(np.array(np.where(group2[1][:, 1] == i))[0])) # 相同材料总数
            num2 += 1
    all_material = list(set(group1[1][:, 1]) | set(group2[1][:, 1]))
    if all_material == []:
        return 0
    similarity2 = num2 / len(all_material)
    similarity1 = num / (len(group1[1]) + len(group2[1]))
    s = (similarity1 + similarity2) * 0.5
    return s

def merge(group1, group2):
    item_num = 1000
    item_square = 250000000
    num = len(group1[1][:, 1]) + len(group2[1][:, 1])
    square = sum(group1[1][:, 2] * group1[1][:, 3]) + sum(group2[1][:, 2] * group2[1][:, 3])
    if num < 1000 and square < item_square:
```

```
47.         return [group1[0] + group2[0], np.append(group1[1], group2[1], axis = 0)]
48.     else:
49.         return []
50.
51. g = [-1]
52. while 1:
53.     temp = g.copy()
54.     similarity_list = []
55.     for i in range(0, len(order_list) - 1): # 重复遍历订单表，计算两两订单相似度
56.         for j in range(i + 1, len(order_list)):
57.             similarity_list.append([order_list[i][0], order_list[j][0], similarity(order_list[i], order_list[j])])
58.             if similarity_list[-1][2] > 0.5:
59.                 for o in order_list: # 找合并的 order
60.                     if similarity_list[-1][0] == o[0]:
61.                         index1 = o
62.                     if similarity_list[-1][1] == o[0]:
63.                         index2 = o
64.                 g = merge(index1, index2)
65.                 if g != []:
66.                     order_list.remove(index1)
67.                     order_list.remove(index2)
68.                     order_list.append(g)
69.                     break_flag = True
70.                     break
71.                 else:
72.                     break_flag = False
73.                     continue
74.             break_flag = False
75.         temp = order_list[i].copy()
76.         del order_list[i]
77.         order_list.append(temp)
78.         if break_flag:
79.             break
80.     if g == []:
81.         break
82.
83. ol = np.array([[0,0,0,0,0]])
84. for i in order_list:
85.     l = len(i[1])
86.     string = '/'
87.     for j in i[0]:
88.         string = string + str(j) + '/'
89.     order = [string] * l
90.     order = np.array(order).reshape(l,-1)
```

```
    temp = np.append(order, i[1], axis = 1)
    ol = np.append(ol, temp, axis = 0).reshape(-1, 5)


name = ['order', 'id', 'material', 'l', 'w']
pd.DataFrame(columns = name, data = ol).to_csv('E:\\19 届华为杯\\2022 年 B 题\\题目\\子问题 2-数据集 B\\batch1.csv')
```

附录 6 排样程序代码

```
IIID=[];
IIIDX=[];
IIIDY=[];
for kk=1:351
    index=find(batch(:,6)==kk);
    dataA=batch(index,[2 4 5]);
    n=size(dataA,1);
    all_data=dataA;
     [~,sort_index]=sort(all_data(:,2),'descend');
    all_data=all_data(sort_index,:);
    all_data_rep=all_data;
    %% 生成条带
     sum=0;
    stripe=[];
    ID={};
    IDX={};
    IDY={};
    while(~isempty(all_data))
        sum=sum+1;
        %%假设第一段为竖直切割
        %%用 stack 去逼近条带，满足多个 stack 宽度类似，长度之和接近 2240，形成多个长条带
        data1=all_data;
        id=[];
        idx=[];
        idy=[];
        %% 排放规则
        %先放入长最大的,则这个条带的长就确定了为 L,id 记录了放入的 item 的编号
            L=data1(1,2);
            all_width=data1(1,3);
            id=[id,all_data(1,1)];
            idx=[idx,L];
            idy=[idy,all_width];
            all_data(1,:)=[];
    %% 先找长与 L 相等的，判断是否能放入
             tmp_index=find(all_data(:,2)==L);
```

```
            if ~isempty(tmp_index)
              data2=all_data;
              tmp1_index=[];
               for i=1:size(tmp_index,1)
                    %判断找到的 tmp_index（i）是否满足条件
               if all_width+data2(tmp_index(i),3)<=1220
                  %能放入则放入
                  all_width=all_width+data2(tmp_index(i),3);
                  id=[id,data2(tmp_index(i),1)];
                  idx=[idx,L];
                  idy=[idy,all_width];
                  tmp1_index=[tmp1_index,tmp_index(i)];
                end
               end
               all_data(tmp1_index,:)=[];
            end
            %% 再找宽与 L 相等的
            tmp_index=find(all_data(:,3)==L);
            if ~isempty(tmp_index)
              data2=all_data;
              tmp1_index=[];
               for i=1:size(tmp_index,1)
           %判断找到的 tmp_index（i）是否满足条件
               if all_width+data2(tmp_index(i),2)<=1220
              %能放入则放入
                  all_width=all_width+data2(tmp_index(i),2);
                  id=[id,data2(tmp_index(i),1)];
                  idx=[idx,L];
                  idy=[idy,all_width];
                  tmp1_index=[tmp1_index,tmp_index(i)];
                end
               end
               all_data(tmp1_index,:)=[];
            end
            %% 下一个长度最长的
          while(all_width<1220)
           %没有与之相等的，就找下一个长度最长的
            m=1220-all_width;
           [ ~,next_ind]=max(all_data(:,3)<m);
           if isempty(next_ind)
              break;
           else
              l=all_data(next_ind(1),2);w=all_data(next_ind(1),3);
              all_width=all_width+all_data(next_ind(1),3);
```

```
80.          end
81.          if all_width>1220
82.             break;
83.          else
84.             id=[id,all_data(next_ind(1),1)];
85.             idx=[idx,l];
86.             idy=[idy,all_width];
87.             all_data(next_ind(1),:)=[];
88.          end
89.          %找长度或者宽度和这个宽度 w 相等的组件放入空隙
90.             all_length=l;
91.             while(all_length<L)
92.                all_length_tmp=all_length;
93.                %% 宽度等于 w 的组件，则放入
94.                 tmp_index=find(all_data(:,3)==w);
95.                  if ~isempty(tmp_index)
96.                  data2=all_data;
97.                  tmp1_index=[];
98.                   for i=1:size(tmp_index,1)
99.                    %判断找到的 tmp_index（i）是否已经用过了
100.                  if all_length+data2(tmp_index(i),2)<=L
101.                     %能放入则放入
102.                     all_length=all_length+data2(tmp_index(i),2);
103.                     id=[id,data2(tmp_index(i),1)];
104.                     idx=[idx,all_length];
105.                     idy=[idy,all_width];
106.                     tmp1_index=[tmp1_index,tmp_index(i)];
107.                  end
108.                 end
109.                 all_data(tmp1_index,:)=[];
110.                end
111.               %% 长度等于 w 的组件,则旋转放入，all_length 加的是其第三列的值 l
112.                tmp_index2=find(all_data(:,2)==w);
113.                if ~isempty(tmp_index2)
114.                data2=all_data;
115.                tmp1_index2=[];
116.                 for i=1:size(tmp_index2,1)
117.                  %判断找到的 tmp_index（i）是否已经用过了
118.                  if all_length+data2(tmp_index2(i),3)<=L
119.                     %能放入则放入
120.                     all_length=all_length+data2(tmp_index2(i),3);
121.                     id=[id,-1*data2(tmp_index2(i),1)];
122.                     idx=[idx,all_length];
123.                     idy=[idy,all_width];
```

```
                    tmp1_index2=[tmp1_index2,tmp_index2(i)];
                end
                end
                all_data(tmp1_index2,:)=[];
                end

            %% 清理已放入的 item
                [~,other,~]=intersect(all_data(:,1),abs(id));
                all_data(other,:)=[];
                if all_length==all_length_tmp %说明此次循环未放入任何组件，则跳出循环
                  break;
                end
            end
        end
        stripe=[stripe,L];
        str=['第',num2str(sum),'个条带组成是',num2str(id),'其长度为',num2str(L)];
        disp(str)
        ID=[ID,{id}];
        IDX=[IDX,{idx}];
        IDY=[IDY,{idy}];
    end
%xlswrite('stripe.xlsx',stripe)
%clear sum;
%% 生成原片
    n=size(stripe,2);
    sum_ban=0;
    num_id=0;
    IID={};
    IIDX={};
    IIDY={};
    while(num_id<n)
        sum_ban=sum_ban+1;
        id=[];
        length=0;
        tmp_ID=[];
        tmp_IDX=[];
        tmp_IDY=[];
        while(length<2440)
            index=find(stripe(1,:)<=(2440-length));
            tmp=find(stripe(index)==-1);
            index(tmp)=[];
            if ~isempty(index)
                length=length+stripe(1,index(1));
```

```
168.                if length<=2440
169.                    id=[id,index(1)];
170.                    tmp_ID=[tmp_ID,cell2mat(ID(index(1)))];
171.                    tmp_IDX=[tmp_IDX,cell2mat(IDX(index(1)))+length-stripe(1,index(1))];
172.                    tmp_IDY=[tmp_IDY,cell2mat(IDY(index(1)))];
173.                    stripe(1,index(1))=-1;
174.                end
175.            else
176.                break;
177.            end
178.        end
179.        IID=[IID,{tmp_ID}];
180.        IIDX=[IIDX,{tmp_IDX}];
181.        IIDY=[IIDY,{tmp_IDY}];
182.        num_id=num_id+size(id,2);
183.         str=['组成第',num2str(sum_ban),'块板的子条目编号为',num2str(id)];
184.         disp(str)
185.    end
186.    IID2=[];
187.    for i=1:sum_ban
188.        IID2=[IID2,cell2mat(IID(i))];
189.        IID2=[IID2,1111111111];
190.    end
191.    IID2=IID2';
192.    IIID=[IIID;IID2];
193.    IIID=[IIID;22222222];
194.
195.    IIDX2=[];
196.    for i=1:sum_ban
197.        IIDX2=[IIDX2,cell2mat(IIDX(i))];
198.        IIDX2=[IIDX2,1111111111];
199.    end
200.    IIDX2=IIDX2';
201.    IIIDX=[IIIDX;IIDX2];
202.    IIIDX=[IIIDX;22222222];
203.    IIDY2=[];
204.    for i=1:sum_ban
205.        IIDY2=[IIDY2,cell2mat(IIDY(i))];
206.        IIDY2=[IIDY2,1111111111];
207.    end
208.    IIDY2=IIDY2';
209.    IIIDY=[IIIDY;IIDY2];
210.    IIIDY=[IIIDY;22222222];
211. end
```

```
IIIID=[IIID,IIIDX,IIIDY];
xlswrite('E:\Jupyter Notebook\Mathematical modeling\19th_Huawei\matlab\data\data5.xlsx',IIIID);
```
