基于启发式搜索和整数规划的 2D/3D 装箱问题安排策略

摘要

中国快递业迅速崛起，构筑了全球最强大、最先进的物流体系之一。其中，包装环节尤为重要，适当的包装耗材能有效保护货物、降低运输风险，并提升物流效能。巨量包裹的特点意味着微小的耗材成本下降也能带来显著经济效益。因此，在不同订单中，选择适宜的包装方案至关重要。

对于问题一，首先，对数据进行预处理，将物体长宽高互换时视为新规格物品，定义**基底**为能放入载体的最小物品，**结合体**为基底的组合。随后，利用组合体与子空间概念，结合**遗传算法**和**整数规划**，将载体的内部物品安排与订单的载体选择两个过程分离，使得载体的选择具有统一的判定标准并得到每个方案所使用的不同载体能够容纳的不同物品数量的向量。再通过定义**箱体利用率**和**装箱得分**，**袋子利用率**和**装袋得分**，**装载利用率**和**装载得分**的方式，在满足容量要求的前提下，针对仅使用箱子、仅使用袋子，以及同时使用箱子和袋子三种情况都选择具有最低得分的方案，即最优方案，详细见附录十。

对于问题二，首先，计算每个订单里每种载体可扩大或缩小的最大体积，排序后对单个订单里前 t 个载体增大，余下的减小，判断是否满足前 t 个载体的可压缩体积大于剩下载体的可扩大体积，不满足的订单中的所有载体按得到的最大缩小值缩小，满足的订单中的所有载体对应尺寸增加，再代入问题一的模型中搜索，得到最终方案的变化。最终，经过优化后耗材尺寸数量**减少了 180** 个，耗材总体积可减少 **5.7%**。

对于问题三，面对货物和耗材都存在柔性或者可挤压的属性时，我们采取耗材可伸展 5% 的假设，在笛卡尔直角坐标系中建立与前两问类似的模型，利用相似算法对前两问模型重新进行求解，得到在柔性情况下的装载方案，最终在经过优化后对于两种柔性材料的数量**减少了 192 个**，耗材总体积可减少 **6.4%**，在全局最优解的前提下具有广泛的应用前景。

关键字：启发式搜索　分支定界法　整数规划　二分答案搜索

一、 问题重述

1.1 背景资料

中国快递业在短短几年内的迅猛发展令人瞩目，其强大的快递物流体系已经成为全球范围内的榜样。然而仅在十年前，中国还是物流成本最昂贵的国家之一。但随着时间的推移，中国已经成功地构建了全球最强大、最先进的快递物流体系之一。在这个庞大的快递体系中，包裹的打包环节显得尤为关键。选择合适的包装耗材对于保护货物、降低运输风险以及提供高效的物流操作至关重要。由于每年的包裹数量巨大，即使是稍微降低每个包裹耗材的成本，也能够带来巨大的经济效益。因此，在包装耗材的选择上，要注重经济性、耐用性和适应不同商品的特点，以最大程度地优化包裹的运输过程。

1.2 需要解决的问题

（1）根据附件 1 中的订单和耗材数据，针对每个订单设计装载方案，可以选择箱子或袋子来装载。目标是尽量减少耗材使用数量，同时在使用数量相同的情况下，降低总体积。最后，给出各种耗材的使用总数和耗材总体积。

（2）基于附件 1 的数据，只对耗材尺寸进行优化，而不改变耗材种数，使得对于问题一，优化后的方案所使用的箱子或袋子数量比之前减少。若耗材数量不改变，则总体积越小越好。最后给出优化后各耗材的具体尺寸、使用次数以及耗材总体积。

（3）上述两个问题都是在货物与耗材都为刚性的前提下，若货物与耗材是柔性的或者可轻微挤压的，装箱方案可能出现变化。根据实际情况，耗材伸展时，长、宽、高都不应超过原尺寸的 5%，在此基础上重新完成问题（1）、(2)。

二、 问题分析

2.1 问题一分析

根据附件一的数据，每个订单涵盖不同种类和数量的物体。对每个订单而言，设计装载方案实质上是一个优化问题，要求以最小的耗材数量为目标，且在耗材数量相同的前提下，追求最小的耗材总体积。首先，我们对数据进行预处理。将每种物体长宽高互换时产生的物体都视为一个新的规格的物品，将至少能放入一个载体的物品视为基底。然后，分三种情况讨论该问题。1. 在完全采用箱子作为包装材料的方案中，通过将不同类型的箱子的长度、宽度和高度与各基底尺寸相除，进行取整运算，以确定在各种箱子内部的基底排列组合，形成所称的“组合体”。每个组合体在置入箱子后，形成三个子空间。接下来，采用朴素搜索方法确定不同箱子可容纳不同种类物品数量的排列组合方

式。随后，引入遗传算法对搜索过程进行优化，并记录各种箱子所容纳不同物品种类数量的排列组合。由此，我们得到了每个方案中使用不同种类箱子所能够容纳的不同组合物品数量向量。在确保所选用的箱子可容纳的物品数量不低于订单所含物品数量的前提下，通过预先定义的“箱体利用率”计算方法，得出装箱得分。最终，从中选择装箱得分最低的方案作为最优解。

2. 在仅使用袋子作为包装材料的情况下，将袋子的高度均设为 1，将问题简化为二维问题。在之前的基础上，每个物体可根据六种不同的布置方式进行放置。通过将每种布置方式的长度调整为原长度加上高度（即 L’=L+H），宽度调整为原宽度加上高度（即 W’=W+H），从而将物体维度转化为二维，每个物体的面积为 S_i。相较于箱子的三维情况，袋子的组合体和子空间变为二维，仅存在两个子空间。并在此基础上重新定义袋子利用率和装袋得分，随后的最优解求解过程与之前只使用箱子的方案相一致。

3. 当同时使用箱子和袋子作为包装材料时，前述步骤保持不变。此时，所定义的载体利用率为“箱体利用率”与“袋子利用率”的总和，而装载得分则是“装箱得分”与“装袋得分”的总和。在保证所选用箱子和袋子的容量不低于订单所需物品数量的前提下，选择装载得分最低的方案作为最优解，后续步骤与仅使用箱子的情况相同。

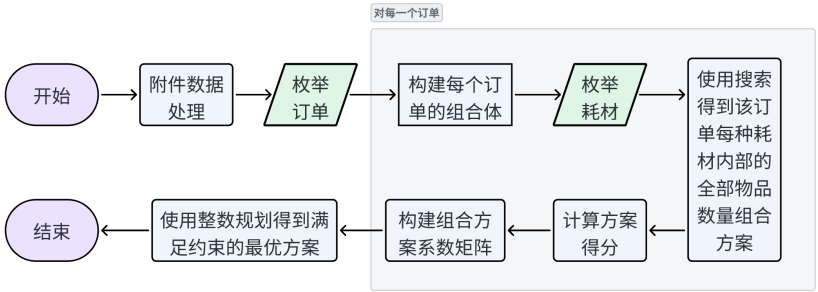

图 1　问题一流程图

2.2 问题二分析

问题二实质上构成了对问题一的优化问题，目标在于通过调整耗材尺寸以获得更优方案。然而，由于缩小载体的尺寸不会改变每个方案中所需各种载体的数量，即方案集本身不会变化而至减小载体的总体积。但扩大载体可能会改变该种载体中物品的摆放方式，即可能改变方案集。所以，载体数量减少的前提条件是方案集发生变化，方案集发生变化的前提条件则是载体的尺寸尽可能增大。为了能留有足够的体积余量去增大剩余类型的载体，我们必须先缩小部分载体的体积。因此，考虑扩大上述排列中的前 t 个载体，余下的减小，并判断每种方案中的箱子在这样更改后是否满足前 t 个载体的可压缩体积大于剩下载体的可扩大体积，不满足则直接缩小所有载体，满足则将扩大的体积分配到指定增加的载体类型中，再带入问题一建立的模型中。

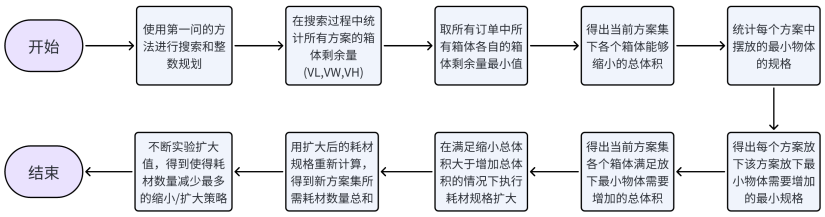

图 2　问题二流程图

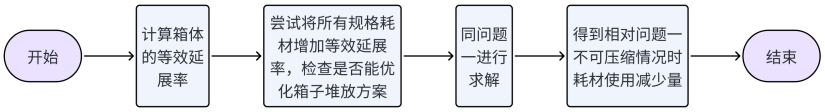

图 3　问题三流程图

2.3 第三问分析

因为题目只给了耗材的伸展率，但是没给货物的伸展率，所以，需要先定义货物的伸展率，然后计算载体在某一边有轻微形变时扩展的长度的比例，以此得到耗材的尺寸变化，重新带入问题二的模型中，的到最终方案。

三、 模型的假设

1. **耗材数量无限，物品数量有限**。
   目标是使用尽可能少的耗材，即使利用率尽可能高。
2. **装载问题是离线的**。
   待装物品已由订单的形式在装载前全部给出。
3. **物品与箱体都是规则物体**。
   给定的物体和箱体都是长方体，给定的袋子可以转换为二维的规则矩形，故给定给定的耗材是弱异质性的。
4. **物品堆放没有限制**。
   物品可以以任意方向放入，物品没有最大承重限制 (即可以任意堆叠)，集装箱没有重心限制 (即可以任意位置堆放)。
5. **物品规格与耗材规格数量级接近**。
   一个箱体在任意单个方向上堆叠数量不超过 100 个，即数量级差距不超过 2。

四、 符号说明

符号	符号说明
(A_i,B_i),i=1,2,3,4	袋子的长、宽
(A_i,B_i,C_i),i=5,6,7,8,9	箱子的长、宽、高
P_i	第 i 种物体的数量
V_i	第 i 种物体的体积
(L_i,W_i,H_i)	第 i 种物体的长、宽、高
(n_i,m_i,r_i)	第 i 种组合分别在长、宽、高方向上的数量
U_i,i=1,2,$\cdots$9	第 i 种耗材的体积
ϕ_i	第 i 种订单的物品集合
$\|\phi_i\|$	订单数
ψ_{ij}=$(c_1,c_2,\cdots,c_k)$	给定 k 种物品在第 i 种耗材中的第 j 种组合向量
Ω_i,i=1,2,$\cdots$	给定物品下第 i 种耗材中物体数量组合方案的集合

五、 问题一模型的建立与求解

5.1 数据预处理

根据题目要求，需要求出只用箱子、只用袋子和二者混合装的三类耗材结果。因此将订单中类型数据分为三类：

1. 全部箱装：剔除不能用箱子装的数据；2. 全部袋装：剔除不能用袋子装的数据；3. 两者同时使用的数据：剔除 1、2 交集的数据。同时，由于物品长宽高可以互换，因此针对一个规格的物品来说，实际上其相当于拥有 6 种不同规格个的变种，我们将不同变种的物品看成是新规格的物品，但是其满足数量和等于长宽高和相等的母规格物品。因此，针对原始订单中的 10000 种不同规格的物品，经过排列组合后形成了 **56990** 个数据。

具体预处理步骤如下：

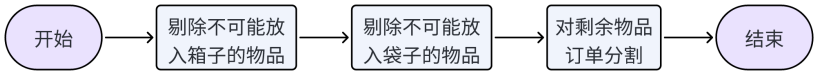

图 4　预处理

5.1.1 空间约束性

箱子数据

对于形成的 56990 个不同规格的数据。其长宽高首先需要满足以下空间约束性，即其本身能够被至少一个规格的箱子装下。

$$\begin{cases} L_{ij} \leq maxL_r \\ W_{ij} \leq maxW_r \\ H_{ij} \leq maxH_r \end{cases} \tag{1}$$

因此，通过对所有规格的数据检索，最终剩下 **23554** 个数据。

袋子数据

对于形成的 56990 个不同规格的数据。其首先需要满足以下空间约束性，即其本身能够被至少一个规格的袋子装下，物品长宽高满足以下约束：

$$\begin{cases} L_{ij} + H_{ij} \leq max(L_r + H_r) \\ W_{ij} + H_{ij} \leq max(W_r + H_r) \end{cases} \tag{2}$$

因此，通过对所有规格的数据检索，最终剩下 **56956** 个数据。

5.2 全使用箱子作为耗材

5.2.1 三维组合体

首先，需要考虑到物体在不同方向上的尺寸互换性。这意味着每种物体都可以按照六种不同的摆放方式在三维空间中进行布局。在为订单中的物体进行装箱规划时，我们采取了一种系统的方法。我们将每种物体的所有可能摆放方式都纳入考虑，尝试将它们适应五种不同尺寸的箱子之中。然后，我们通过排除那些无法放置于任何箱子中的摆放方式，来确定可行的摆放方案，这些方案被称为基底。为了进一步优化装箱策略，我们将各个基底的尺寸与不同箱子的长、宽和高进行比较，并进行整数运算以确定每种基底在各种箱子中的排列组合方式。这些排列组合构成了不同的物体布局，通常被称为组合体[2]。为了帮助理解，以下图示展示了一个示例，演示了如何将六个物体合理地组合为一个组合体，

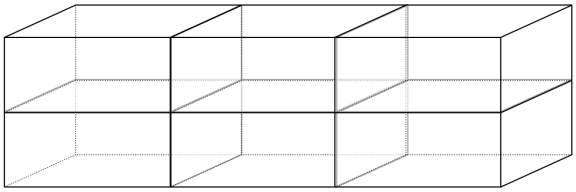

图 5　组合体

5.2.2 子空间

在组合体被装入箱子后，箱子将被划分为三个不同的子空间，如下所示：

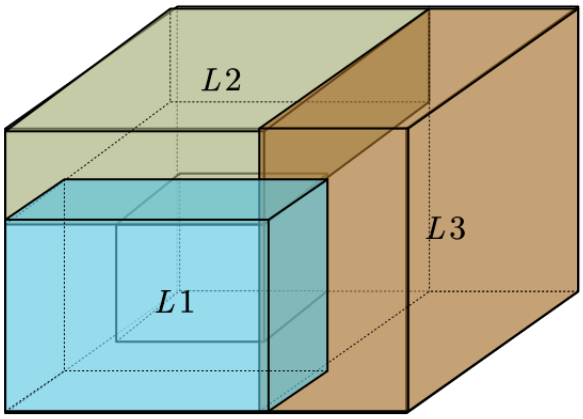

图 6　分成三个子空间

5.2.3 朴素搜索

当某组合体的尺寸（即其长、宽或高）小于子空间的对应尺寸时，可认为此组合体不适合放置于该子空间内。如果连最小的组合体（即基底组合体）也不适合，我们可以确定该子空间已达到最大装载容量[3]。当所有子空间都无法装载任何新的组合体时，整个箱体则被视为已完全装载，接下来的物品将被放入新的箱子中。因此，我们获得了第 i 种载体（其中 i 的取值为 5、6、7、8、9，在本小节后续部分保持不变，即只关注箱子）中能够容纳 k 种不同种类物品的数量的 j 种组合：$\psi_{ij} = (c_1, c_2, \ldots, c_k)$

5.2.4 遗传算法

接着，我们使用遗传算法降低上述搜索方式的复杂度[1] 。针对某一订单, 我们定义简单遗传模型 $SGA = (C, E, P_0, \Phi, \tau, T)$ 如下：

参数	参数设置
C(个体编号方法)	基因位标记当前组合体旋转方向，共 6 个基因
染色体表示当前订单每一个组合体的旋转方向	
E(个体适应度评价函数)	$E = sum_{i=1}^{k}(C_i + 1 - C_i \cdot V_i/U)$
P_0(初始群体构造)	枚举最大组合体的旋转方向，其余方向随机
Φ (选择算子)	记两个亲代的适应度为 E_1,E_2, 则某一集因位选取第一个亲代基因的概率为 $\eta = E_1/(E_1 + E_2)$
τ (变异算子)	采取单点变异，使用混沌序列得到第 $n+1$ 位基因的概率为 $x_{n+1} = 4 \cdot x_n \cdot (1-x_n)$
T(终止条件)	繁殖 200 代

5.2.5 整数规划

接下来，我们记录第 i 种箱子能够容纳的不同种类物品数量的组合共有 N_i 种。在这些组合中，我们用 Y 表示 N_i 中的最大值，即 $Y = \max\{N_i\}$。记 d_{ij} 为第 i 种箱子第 j 种方案的选择数量，其系数矩阵表示为一个 5×Y 的矩阵 $Num1$（因为只有五种不同的箱子），即：

$$Num1 = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1N(1)} & 0 & 0 \\ d_{21} & d_{22} & \cdots & \cdots & \cdots & d_{iN(2)} \\ \vdots & & & & & \vdots \\ d_{51} & \cdots & d_{5N(i)} & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

其中，对于第 i 种箱子，超过 N_i 种方案的选择数量都被设为零。需要强调的是，d_{ij} 在其中的位置只是为了更好地阐述问题，并不是代表实际位置，这种安排仅用于更便于理解。

于是，每个方案所使用的不同种箱子能够容纳的不同种组合的物品数量的向量为:$d_{ij} \cdot \psi_{ij} = (d_{ij} \cdot c_1, d_{ij} \cdot c_2, \cdots, d_{ij} \cdot c_k)$

同时，令 sum_i 表示第 i 种箱子的使用数量，通过下式可得每种箱子的使用数量：

$$sum_i = \sum_{j=1}^{N_(i)} d_{ij}, i = 1, 2, \cdots, 5 \tag{4}$$

随后，我们引入箱体利用率的概念：该订单所有物品的体积之和与选定的箱子的总体积之比，即：

$$\eta 1 = \sum_{k=1}^{|\Omega_i|} V_k / \sum_{i=1}^{5}(U_i \cdot sum_i) \tag{5}$$

因此，我们可以引入一个装箱得分（score1）的概念：该得分由所使用的箱子数量加上一减去箱体利用率所得，即：

$$score1 = \sum_{i=1}^{5}(1 - sum_i) + \eta 1 \tag{6}$$

显而易见的是，得分越低则意味着方案越为优越。

因为最终方案所使用的箱子能够容纳的物品数量必须超过订单中的物品数量，所以：

$$\sum_{i=1}^{5}\sum_{j=1}^{Y} d_{ij}\cdot\phi_{ij} - Q \geq 0 \tag{7}$$

其中的 $Q=(P1,P2,\cdots,P_k)$，而 $P1,P2,\cdots,P_k$ 表示每个订单中的物体数量。

综上，(6) 式为我们的目标函数，(7) 式为我们的约束条件，构建整数规划并使用分支定界法求解，经统计，附件一中的全部订单共计需要箱子数量的数据如下（每个订单具体箱子数量见附录十）。

纸箱	数量 (个)	总体积
普通 1 号自营纸箱	788	
普通 2 号自营纸箱	870	
普通 3 号自营纸箱	861	23110332000
普通 4 号自营纸箱	2845	
普通 5 号自营纸箱	279	

5.3 全使用袋子作为耗材

5.3.1 子空间

鉴于所有类型的袋子的高度均为 1，因此我们可以将这个讨论视为一个二维问题。基于之前的讨论，每个物体可以按照六种不同的布置方式来放置。通过将每个布置方式的长度设置为原来的长加上高 (即 L’=L+H)，宽设置为原来的宽加高 (即 W’=W+H)，此时能够将物体的维度转变为二维，同时第 i 个物体的面积为 S_i。对比箱子的三维情况，袋子中的组合体也变成了二维的概念，袋子分成的子空间也是二维的，且只有两个：

5.3.2 二维组合体

袋子被视为已完全装载的条件也和箱子一样，因此，以同样的方法可得因此，我们获得了第 i 种载体（其中 i 的取值为 1，2，3，4，在本节后续部分保持不变，即只关注袋子）中能够容纳 k 种不同种类物品的数量的 j 种组合：由此，得到了第 i 种袋子中能装下的 k 种不同种类物品的数量的 j 种组合：$\psi_{ij}=(c_1,c_2,\ldots,c_k)$

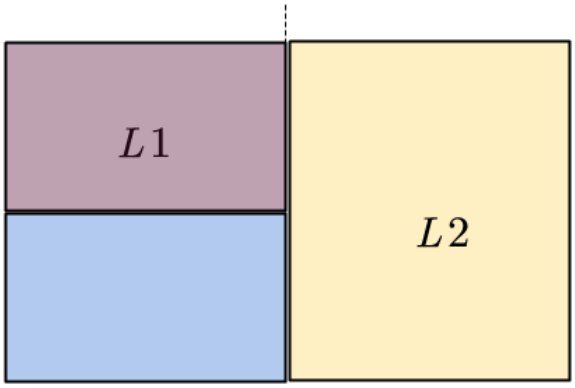

图 7　分成两个子空间

5.3.3 朴素搜索

同样的，我们记录第 i 种袋子能够容纳的不同种类物品数量的组合共有 N_i 种。在这些组合中，我们用 Y 表示 N_i 中的最大值，即 $Y = \max\{N_i\}$。记 d_{ij} 为第 i 种箱子第 j 种方案的选择数量，其系数矩阵表示为一个 4×Y 的矩阵 $Num2$（因为只四种不同的袋子），即：

$$Num2 = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1N(1)} & 0 & 0 \\ d_{21} & d_{22} & \cdots & \cdots & \cdots & d_{iN(2)} \\ \vdots & & & & & \vdots \\ d_{51} & \cdots & e_{5N(i)} & 0 & 0 & 0 \end{bmatrix} \tag{8}$$

其中，对于第 i 种袋子，超过 N_i 种方案的选择数量都被设为零。需要强调的是，d_{ij} 在其中的位置只是为了更好地阐述问题，并不是代表实际位置，这种安排仅用于更便于理解。

于是，每个方案所使用的不同种袋子能够容纳的不同种组合的物品数量的向量仍然为:$d_{ij} \cdot \psi_{ij} = (d_{ij} \cdot c_1, d_{ij} \cdot c_2, \cdots, d_{ij} \cdot c_k)$

同时，令 sum_i 表示第 i 种袋子的使用数量，通过下式可得每种箱子的使用数量：

$$sum_i = \sum_{j=1}^{N_(i)} d_{ij}, i = 1, 2, \cdots, 5 \tag{9}$$

随后，我们引入袋子利用率的概念：该订单中所有物品在二维平面上的面积之和与所选袋子的体积（由于袋子的高度为 1，因此体积等于面积）之比，即：

$$\eta 2 = \sum_{k=1}^{|\Omega_i|} S_k / \sum_{i=1}^{4} (U_i \cdot sum_i) \tag{10}$$

因此，我们可以引入一个装袋得分（score2）的概念：该得分由所使用的袋子数量

加上一减去袋子利用率所得，即：

$$score2 = \sum_{i=1}^{4}(1 - sum_i) + \eta 2 \tag{11}$$

显而易见的是，得分越低则意味着方案越为优越。

因为最终方案所使用的袋子能够容纳的物品数量必须超过订单中的物品数量，所以：

$$\sum_{i=1}^{4}\sum_{j=1}^{Y} d_{ij} \cdot \phi_{ij} - Q \geq 0 \tag{12}$$

其中的 $Q = (P1, P2, \cdots, P_k)$，而 $P1, P2, \cdots, P_k$ 表示每个订单中的物体数量。

综上，(12) 式为我们的目标函数，(13) 式为我们的约束条件。

构建整数规划并使用分支定界法求解，经统计，附件一中的全部订单共计需要袋子的具体数目如下（每个订单具体袋子数目见附录十）：

袋子	数量 (个)	总体积
普通 1 号袋	138082500	724546500
普通 2 号袋	231675000	
普通 3 号袋	291852000	
普通 4 号袋	62937000	

5.4 箱子和袋子同时使用

箱子和袋子同时使用对于我们前面建立的模型没有任何影响，仍然可以用相同的方法得到第 i 种载体（这时 i 取 1 到 9，在本节后续部分保持不变，即同时关注箱子和袋子）种能够容纳 k 种不同种类物品的数量的 j 种组合：$\psi_{ij} = (c_1, c_2, \ldots, c_k)$

同样的，我们仍然记录第 i 种载体能够容纳的不同种类物品数量的组合共有 N_i 种，用 Y 表示 N_i 中的最大值，即 $Y = \max\{N_i\}$。d_{ij} 仍为第 i 种载体第 j 种方案的选择数量，只是此时的系数矩阵表示为一个 9×Y 的矩阵 $Num.$

每个方案所使用的不同种袋子能够容纳的不同种组合的物品数量的向量仍然为:$d_{ij} \cdot \psi_{ij} = (d_{ij} \cdot c_1, d_{ij} \cdot c_2, \cdots, d_{ij} \cdot c_k)$

同时，令 sum_i 表示第 i 种载体的使用数量，通过下式可得每种箱子的使用数量：

$$sum_i = \sum_{j=1}^{N_(i)} d_{ij}, i = 1, 2, \cdots, 9 \tag{13}$$

随后，我们引入载体利用率的概念：即箱体利用和袋子利用率之和：

$$\eta = \eta 1 + \eta 2 \tag{14}$$

因此，我们可以引入一个载体得分（score）的概念：该得分由箱体得分和袋子得分相加得到，即：

$$score = score1 + score2 \tag{15}$$

显而易见的是，得分越低则意味着方案越为优越。

因为最终方案所使用的载体能够容纳的物品数量必须超过订单中的物品数量，所以：

$$\sum_{i=1}^{9}\sum_{j=1}^{Y} d_{ij} \cdot \phi_{ij} - Q \geq 0 \tag{16}$$

其中的 $Q = (P1, P2, \cdots, P_k)$，而 $P1, P2, \cdots, P_k$ 表示每个订单中的物体数量。

综上，(15) 式为我们的目标函数，(16) 式为我们的约束条件。构建整数规划并使用分支定界法求解，经统计，附件一中的全部订单如下（各个订单的具体使用结果见附录十。）：

袋子	数量 (个)	总体积
普通 1 号袋	2907	
普通 2 号袋	3024	
普通 3 号袋	1812	
普通 4 号袋	168	
普通 1 号自营纸箱	0	788818500
普通 2 号自营纸箱	0	
普通 3 号自营纸箱	0	
普通 4 号自营纸箱	0	
普通 5 号自营纸箱	15	

六、 问题二模型的建立与求解

6.1 全使用箱子作为耗材

针对箱子，可以建立如下空间直角坐标系：其中，X0、Y0、Z0 分别代表该箱子的

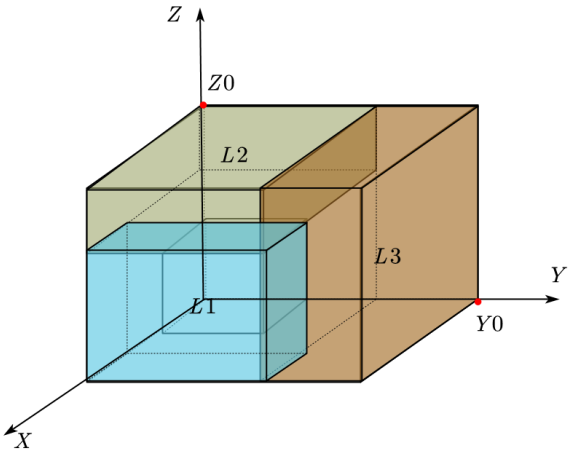

图 8　箱子坐标系

长、宽、高。

根据第一问的分析可知，在将组合体放入箱子时，始终追求将其放入最大可能容纳的位置。换言之，在任意子空间内，首先放入的组合体尺寸大于随后放入的组合体，而且均位于子空间的左上角。再根据组合体的尺寸大小，我们可以推导出所有不同方向的子空间（上方、前方、右方）中尺寸最大的组合体在相应方向上的坐标，即可以知道每个箱子里沿着上方、前方、右方的物品放置情况[4]：

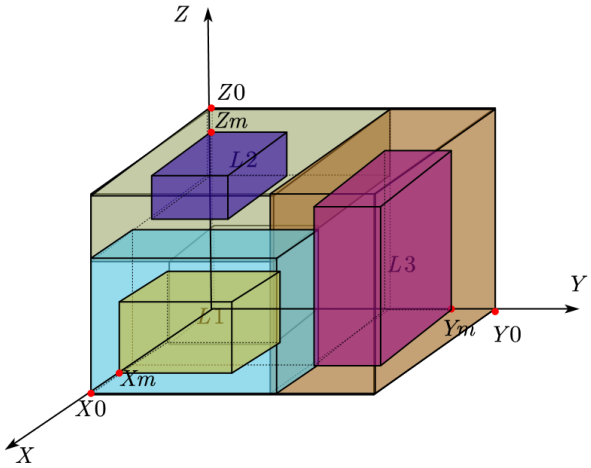

图 9　沿不同方向的放置情况

其中，X_m、Y_m、Z_m 分别表示在不同方案下的各种箱子在放入所有物品后，沿着该箱子长、宽、高对应的最大坐标。

所以，沿着长、宽、高方向的可压缩的空间 VL,VW,VH 分别为：

$$\begin{cases} VL = X0 - Xm \\ VW = Y0 - Ym \\ VH = Z0 - Zm \end{cases} \tag{17}$$

所以可得在问题一中得到的最优方案下第 j 种载体（j=5…9，在本节后续部分保持不变，即只关注箱子) 的最大可压缩体积为：

$$DV_j = m_j \cdot [VL_j \cdot Y0 \cdot Z0 + VW_j \cdot (X0 - \triangle L_j) \cdot Z0 + VH_i \cdot (X0 - VL_j)(Y0 - VW_j)] \tag{18}$$

因此，在第 i 个订单（共 5133 个）中，第 j 种载体的可压缩长度、宽度和高度分别可表示为 $\triangle L_{ij}$、$\triangle W_{ij}$ 和 $\triangle H_{ij}$。通过取这些值的最小值，我们能够得到第 j 种载体在长度、宽度和高度方向上的最大可压缩值：

$$\begin{cases} \triangle L_j = \min_{i=1}^{5133} \triangle L_{ij} \\ \triangle W_j = \min_{i=1}^{5133} \triangle W_{ij} \\ \triangle H_j = \min_{i=1}^{5133} \triangle H_{ij} \end{cases} \tag{19}$$

同时，记录第 i 号订单中放置于第 j 种载体的最小物品的尺寸为 $(ML_{ij}, MW_{ij}, MH_{ij})$，于是我们可以知道所有订单中的最小物品的尺寸，所以箱子需要延展的最小尺寸，就是该物体的尺寸，即：$(\triangle L_{ij}, \triangle W_{ij}, \triangle H_{ij}) = (ML_{ij} - VL_{ij}, MW_{ij} - VW_{ij}, MH_{ij} - VH_{ij}$。再有上式 16，可知第 j 种载体的最小可延展体积为：

$$AV_j = m_j \cdot [\triangle L_j \cdot Y0 \cdot Z0 + \triangle W_j \cdot (X0 - \triangle L_j) \cdot Z0 + \triangle H_i \cdot (X0 - \triangle L_j)(Y0 - \triangle W_j)] \tag{20}$$

故第 j 种载体可压缩/扩大的变化体积记为 (DV_i, AV_i)，然后按照 DV_i 降序排列，得到 (DV_1, AV_1)，(DV_2, AV_2)，$(DV_3, AV_3)…(DV_9, AV_9)$。

由于缩小载体的尺寸不会改变每个方案中所需各种载体的数量，即方案集本身不会变化而至减小载体的总体积。但扩大载体可能会改变该种载体中物品的摆放方式，即可能改变方案集。我们的优化目标是尽可能减少载体的使用数量，约束条件是不能改变载体的总体积。所以，载体数量减少的前提条件是方案集发生变化，方案集发生变化的前提条件则是载体的尺寸尽可能增大。而由于有约束条件的存在，我们必须先缩小部分载体的体积，才能留有足够的体积余量去增大剩余类型的载体。而由于同一种载体不能既缩小又增加，故我们希望尽可能少的类型的载体尺寸缩小，其余载体尺寸扩大。因此，考虑扩大上述排列中的前 t 个箱子，余下的减小，并判断每种方案中的箱子在这样更改后是否满足该条件：

$$\sum_{t}^{i=1} DV_i \geq \sum_{9}^{t+1} AV_i (t = 1 \ldots 8) \tag{21}$$

若缩小前 8 种载体仍然无法使得第 9 种载体扩大的体积达到 $(\triangle L_9, \triangle W_9, \triangle H_9)$ 方案集发生变化，则将这些箱子的体积全部缩小即可。

若有能够发生变化的条件，则将扩大的体积分配到指定增加的箱体类型上：

$$(X_j', Y_j', Z_j') = (X_j + \triangle L_j, Y_j + \triangle Y_j, Z_j + \triangle Z_j)(K = 5 \ldots 9) \tag{22}$$

之后，重新带入问题一中建立的模型进行搜索，得到优化后的每种耗材的具体尺寸如下：

耗材	长	宽	高
普通 1 号自营纸箱	165	120	55
普通 2 号自营纸箱	200	140	70
普通 3 号自营纸箱	352	302	302
普通 4 号自营纸箱	270	200	90
普通 5 号自营纸箱	300	200	170

6.2 全使用袋子作为耗材

而对于袋子的处理方法本质仍然相同，只是要考虑的问题的维度从三维变成二维而已，此时只需要建立平面直角坐标系而非空间直角坐标系如下：

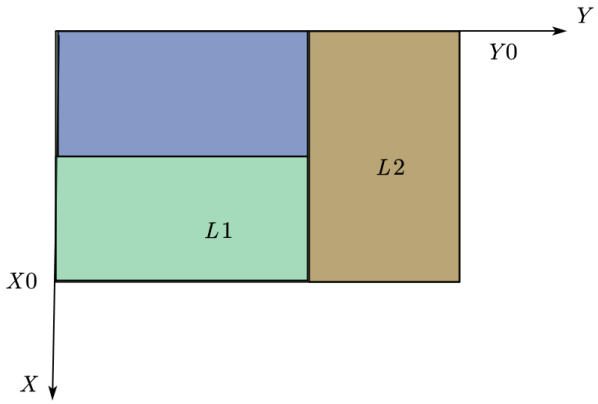

图 10　袋子坐标系

同理，在不同方案下的各种袋子在放入所有物品后，沿着该袋子长、宽对应的最大坐标仍为 Y_m,X_m:

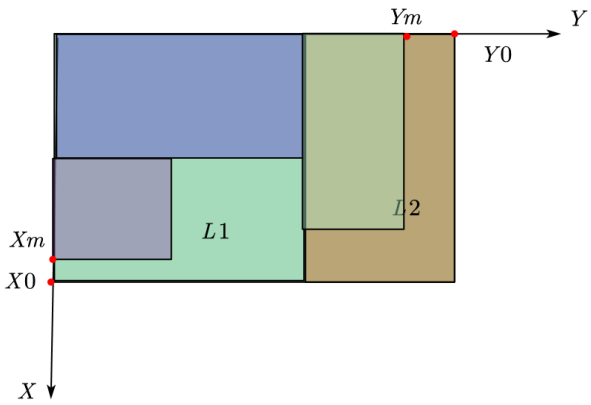

图 11　沿不同方向的放置情况

所以，沿着长、宽方向的可压缩的空间 X_2,Y_2 分别为：

$$\begin{cases} X_2 = X0 - Xm \\ Y_2 = Y0 - Ym \end{cases} \tag{23}$$

那么，每个订单里不同袋子压缩后的面积

接下来，综合比较所有订单中的每种袋子在压缩后的体积如下：

耗材	长	宽	高
普通 1 号袋子	208	148	1
普通 2 号袋子	300	250	1
普通 3 号袋子	400	330	1
普通 4 号袋子	450	420	1

6.3 箱子和袋子同时使用

综上，我们得到了全部订单每种耗材所需要的数量 $M = (m_1, m_2, \ldots, m_q)$，所以每种方案中的第 i 种载体使用数量和耗材总体积的部分数据如下，详细见附录?

6.4 结果分析

最终，经统计，优化后耗材尺寸数量减少了 180 个，耗材总体积可减少 **5.7%**, 取得了较好的效果。

袋子	数量 (个)	总体积
普通 1 号袋	726	
普通 2 号袋	5205	
普通 3 号袋	1812	
普通 4 号袋	168	
普通 1 号自营纸箱	0	549922848
普通 2 号自营纸箱	0	
普通 3 号自营纸箱	15	
普通 4 号自营纸箱	0	
普通 5 号自营纸箱	0	

七、 问题三模型的建立与求解

7.1 问题一的重解

已知耗材延展量不超过原先尺寸的 5%，而不知道货物的压缩率。为此，我们定义货物的平均压缩率 ϵ，在挤压时箱体在某一方向的尺寸伸展的等效长度为 $L' = 105\%/(1-\epsilon)$，考虑实际情况，本文取 $\epsilon = 3\%$ 因此我们的策略是假设载体已经延展了这样的尺寸。能够这样假设的理由是，如果安排方案中物品与载体发生挤压，则证明这样的延展是有效的；若没有发生挤压，则说明安排方案没有发生改变，故对原方案集得到的结果不发生影响。结果如下。

7.1.1 全使用箱子作为柔性耗材

基于问题三柔化假设，通过完善问题一模型，我们通过整数规划求解得到订单结果。经统计，附件一中的全部订单共计需要箱子数量如下（具体订单数据详见附录十）:

7.1.2 全使用袋子作为柔性耗材

基于问题三柔化假设，通过完善问题一模型，我们通过整数规划求解得到订单结果。经统计，附件一中的全部订单共计需要袋子数量如下（具体订单数据详见附录十）:

袋子	数量 (个)	总体积
普通 1 号自营纸箱	884	20454536000
普通 2 号自营纸箱	1433	
普通 3 号自营纸箱	947	
普通 4 号自营纸箱	2218	
普通 5 号自营纸箱	161	

袋子	数量 (个)	总体积
普通 1 号袋	163542500	655884500
普通 2 号袋	24922500	
普通 3 号袋	213444000	
普通 4 号袋	29673000	

7.1.3 箱子和袋子同时作为耗材

基于问题三柔化假设，通过完善问题一模型，我们通过整数规划求解得到订单结果。经统计，附件一中的全部订单共计需要两种耗材的具体数量如下（具体订单数据详见附录十）：

7.2 问题二的重解

在此假设下，对问题二的重解于对问题一的重解总体上没有区别，有一点不同的是由于载体和物体可以压缩，因此可以缩小的总体积增加了，故将 (24) 式重写如下：

$$105\%/(1-\epsilon)\sum_{t}^{i=1} DV_i \geq \sum_{9}^{t+1} AV_i(t=1\ldots 8) \tag{24}$$

再依据对问题一的重解继续求解即可。

7.2.1 优化后柔性尺寸表

通过对问题二重解，我们得到了优化后的所有柔性耗材尺寸表如下：

同时，得到了优化后两种柔性耗材同时使用的总数和耗材的总体积表如下：

袋子	数量 (个)	总体积
普通 1 号袋	3443	
普通 2 号袋	3229	
普通 3 号袋	1189	
普通 4 号袋	65	
普通 1 号自营纸箱	0	574959500
普通 2 号自营纸箱	0	
普通 3 号自营纸箱	0	
普通 4 号自营纸箱	0	
普通 5 号自营纸箱	0	

耗材	长	宽	高
普通 1 号袋子	200	140	1
普通 2 号袋子	300	250	1
普通 3 号袋子	400	330	1
普通 4 号袋子	450	420	1
普通 1 号自营纸箱	165	120	55
普通 2 号自营纸箱	200	140	70
普通 3 号自营纸箱	360	310	310
普通 4 号自营纸箱	270	200	90
普通 5 号自营纸箱	300	200	170

7.3 结果分析

最终在经过优化后对于两种柔性材料的数量减少了 192 个，耗材总体积可减少 **6.4%**，耗材总体积进一步减少，说明本模型存在实际有效的推广前景。

袋子	数量 (个)	总体积
普通 1 号袋	717	
普通 2 号袋	5955	
普通 3 号袋	1189	
普通 4 号袋	65	
普通 1 号自营纸箱	0	150994261
普通 2 号自营纸箱	0	
普通 3 号自营纸箱	0	
普通 4 号自营纸箱	0	
普通 5 号自营纸箱	0	

八、 模型的评价

8.1 模型的优点

1. 基于搜索而非贪心策略建立的模型与全局最优解有很高的接近度。该递归算法能灵应用于问题的自然分解结构，能够建立不同的解决方法，可以直接反映问题的分解方式。

2. 将载体的内部的物品安排与订单的载体选择两个过程分离，使得载体的选择具有统一的判定标准，从而实现袋子与箱子两种维度的载体统一考虑，极大提升了模型的可拓展性。

3. 基于分支定界法实现的整数规划算法较为成熟。可以通过限制搜索空间，有效地探索不同的解空间，

4. 基于遗传算法构建模型，适用于大规模、复杂的组合优化问题，能够处理巨大的搜索空间，同时通过迭代进化寻找全局最优解，具有一定的全局搜索能力。

8.2 模型的缺点

1. 无论是朴素搜索，递归，遗传算法还是分支定界法，都需要大量的计算资源，因此在大规模空间数据集上的计算成本和计算复杂度高。

九、 模型的推广

通过建立基于 GA-BF 袋箱装载模型，对二维和三维的个体和总体袋箱优化装载，从而使相关耗材有效减少。在物流业蓬勃发展的现在，对世界范围内的物流业在节约耗材方面都有不小的行业前景[1]。

十、参考文献

[1] 陈胜达. 《基于遗传和递归的装箱算法研究》. 硕士, 厦门大学, 2007.

[2] 韩运实. 《装箱问题方法研究及其集成应用》. 硕士, 中国海洋大学, 2004.

[3] 江宝钏, 和熊伟清. 《一种求解三维集装箱装箱问题的混合遗传算法》. 计算机工程与应用, 期 26 (2007 年): 200-202+222.

[4] 张德富, 彭煜和张丽丽. 《求解三维装箱问题的多层启发式搜索算法》. 计算机学报 35, 期 12 (2012 年): 2553–61.

附录一　箱子数据

```
clc; close all; clear;
%只装箱子排列组合
% 读取订单数据
orderData = xlsread('E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\处理数据\工作簿1.xlsx');

% 读取箱子数据
boxData = xlsread('E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\原始数据\箱子数据.xlsx');

% 数据预处理
orderDimensions = orderData(:, 2:4); % 提取物品的长宽高数据

% 初始化数据容器
Data1 = []; % 全使用箱子装的物品
allData=[];

%%
% 只装袋子
for i = 1:size(orderData, 1)
    itemDimensions = orderDimensions(i, :);

        % 将订单号拼接到每个排列组合数据中
            % 生成所有可能的长宽高排列
    allCombinations = generateCombinations(itemDimensions);
    orderNumbers = repmat([orderData(i, 1),orderData(i,5)], size(allCombinations, 1), 1);
    combinedData = [orderNumbers, allCombinations];

    % 将当前订单的排列组合数据添加到allData中
    allData = [allData; combinedData];
end
%%
% 遍历所有排列组合数据
for i = 1:size(allData, 1)
    orderNumber = allData(i, 1:2); % 提取订单号
    dimensions = allData(i, 3:end); % 提取排列组合的长宽高
    canUseBox = false;

    % 遍历袋子数据，检查是否可以使用箱子装
    for k = 1:size(boxData, 1)
        boxDimensions = boxData(k, 3:5);
        if all(dimensions <= boxDimensions)
           canUseBox = true;
           dimensions(:,4)=boxData(k,1);
           dimensions(:,5:7)=boxData(k,3:5);
            break;
```

```
        end
    end

    % 判断是否需要进行存储
    if canUseBox
        % 将订单号和排列组合的长宽高存储到Data1中
        combinedData = [orderNumber, dimensions];
        Data1 = [Data1;combinedData];
    end

end
%%

% 获取唯一的订单号
order_numbers = unique(Data1(:, 1));

% 指定保存文件夹路径
output_folder = 'E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\处理数据\所有订单箱子数据';

% 遍历订单号，保存数据
for i = 1:length(order_numbers)
    order_number = order_numbers(i);
    order_data = Data1(Data1(:, 1) == order_number, :);

    output_filename = fullfile(output_folder, sprintf('%d.csv', order_number));
    dlmwrite(output_filename, order_data, '\t');
end

% 排列组合函数，用于生成所有可能的长宽高排列
function combinations = generateCombinations(dimensions)
    combinations = perms(dimensions);
    combinations = unique(combinations, 'rows');
end
```

附录 二　所有订单的袋子数据

```
clc; close all; clear;
%% 袋子所有订单一次性处理
clc;
close all;
clear;

inputFolderPath =
    'E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\处理数据\所有订单袋子数据\';
outputFolderPath =
```

```
    'E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\处理数据\所有订单袋子装载数据\';

% 获取所有订单数据文件名
orderDataFiles = dir(fullfile(inputFolderPath, '*.csv'));

% 提取文件名
fileNames = {orderDataFiles.name};

% 提取文件名中的数字部分
fileNumbers = cellfun(@(name) sscanf(name, '%d'), fileNames);

% 对文件名按照数字部分排序
[sortedFileNumbers, sortedIndices] = sort(fileNumbers);
sortedFileNames = fileNames(sortedIndices);
sortedFileNames = sortedFileNames';
%%
 for fileIdx = 1:length(sortedFileNames)
%  for fileIdx = 1:2
    recordedData=[];
    orderRows=[];
    dataStruct = [];
   orderDataFileName = sortedFileNames{fileIdx};
   orderDataFilePath = fullfile(inputFolderPath, orderDataFileName);

   % 从文件名中提取编号部分
   [~, orderFileName, ~] = fileparts(orderDataFileName);
   orderNumber = fileNumbers(sortedIndices(fileIdx));
   orderDataFile=fullfile(orderDataFilePath);

      orderData = readtable(orderDataFile);
      orderData = table2array(orderData);
    % 读取箱子数据
      bagData =
          xlsread('E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\原始数据\袋子数据.xlsx');
% 提取规格和数量信息
uniqueOrders = unique(orderData(:, 1));
results = [];
Quantity=[];
long=[];
wid=[];
hig=[];

% 遍历每个订单
for i = 1:length(uniqueOrders)
   order = uniqueOrders(i);
   orderRows = orderData(orderData(:, 1) == order, :);
```

```
    % 提取订单数据中规格的数量
    orderQuantities = orderRows(:, 2);
    dimensions = orderRows(:, 3:5);

    % 遍历每个规格
    for j = 1:size(orderRows, 1)
        quantity = orderQuantities(j);
        dims = dimensions(j, :);
        V=dims(1) * dims(2) * dims(3);
        length1=dims(1,1);
        width1=dims(1,2);
        high1=dims(1,3);

        for k=1:size(bagData, 1)
        % 计算最大矩形体数量
        boxDims = bagData(k, 3:5);
        V_box= prod(quantity(:,:)*bagData(k,3:5));


        maxlong=floor(boxDims(:,1)/length1);
        maxwidth=floor(boxDims(:,2)/width1);
%         maxhigh=floor(boxDims(:,3)/high1);


        long=[long;maxlong];
        wid=[wid;maxwidth];
%         hig=[hig;maxhigh];
        maxQuantity = ([long, wid]);
        end



    end
end
h = size(orderRows,1); % 替换为您的需求，指示要分割的批次数量
t= size(orderRows,1);
batchSize = size(maxQuantity, 1) / h; % 计算每个批次的大小
y=h;
for i = 1:h
    startIndex = (i - 1) * batchSize + 1; % 计算起始索引
    endIndex = i * batchSize; % 计算结束索引

    variableName = ['data' num2str(i)]; % 构造变量名
    dataStruct.(variableName) = maxQuantity(startIndex:endIndex, :); % 将数据赋值给相应变量

end
```

```
% 初始化记录的变量
recordedData = [];
fields = fieldnames(dataStruct);

% 遍历每个 h 和 p 组合
for i = 1:length(fields)
 fieldName = fields{i};
h = str2double(fieldName(5:end)); % 假设字段名形如 "data10", 提取从第 5 个字符到末尾的部分

    datah = dataStruct.(fields{i});

    for p = 1:size(datah, 1)
        rowValues = datah(p, :);

        % 检查行数据是否全部非零
        if all(rowValues ~= 0)
            recordedData = [recordedData; h, p, rowValues];
        end
    end
end

% 根据 recordedData 中的 h 和 p 值检索相关数据
newData = [];
for l = 1:size(recordedData, 1)
    hValue = recordedData(l, 1);
    pValue = recordedData(l, 2);

    orderDataValues = orderData(hValue, 1:5);
    boxDataValues = bagData(pValue, 3:5);
    newData = [newData; orderDataValues, boxDataValues, recordedData(l, 3:end)];
end
%%

% 计算并添加新的列数据到 newData 中
newData(:,11)=1;
l1= newData(:,9);%长的个数
l2=newData(:,10);%宽的个数
l3=newData(:,11);%高的个数
numl123=l1 .* l2 .* l3 ;%袋子不考虑高
newData(:,12)=numl123; %一个箱子中一共能摆放的数量
for i = 1: size(newData,1)
    if newData(i,12) >= newData(i,2)
       newData(i,13) = newData(i,2); %实际用的箱子装载的个数
    else
        newData(i,13) =newData(i,12);
    end
end
```

```
newData(:, 14) = newData(:, 6) .* newData(:, 7) .* newData(:, 8) ;%箱子单个体积
newData(:, 15) = newData(:, 3) .* newData(:, 4) .* newData(:, 5) .* newData(:,13)
    ;%货物实际总体积
newData(:, 16) = ceil(newData(:, 13) .\ newData(:, 2)); % 箱子数量，向上取整，如果有余数则加 1
newData(:, 17) = newData(:, 14) .* newData(:,16) ;%使用箱子总体积
newData(:,18) = 1-(newData(:,17) .\ newData(:,15))+ newData(:,16); % 得分 =
    1-（实际装载的总体积/使用箱子的总体积）+箱子个数

%%
% 将 newData 按照第 12 列数据从大到小排序
sortedData = sortrows(newData, -11);

% 将排序后的数据保存为 needdata
needdata = sortedData;

%%
% 计算第二到第四列的和作为特征
feature_sums = sum(needdata(:, 3:5), 2);
%%
% 初始化一个列向量，用于存储类别信息
categories = zeros(size(needdata, 1), 1);

% 根据特征和计算类别
unique_sums = unique(feature_sums);
numCategories = numel(unique_sums);

for i = 1:numCategories
    matchingRows = feature_sums == unique_sums(i);
    categories(matchingRows) = i;
end
% 将类别信息添加到原数据中
data_with_categories = [needdata, categories];
OOdata=[needdata(:,15),needdata(:,9),needdata(:,10),needdata(:,11),needdata(:,3),needdata(:,4),needdata(:,5),da
% recordedData=[];

    outputFileName = sprintf('第%d号装载数据.xlsx', orderNumber);
    outputFilePath = fullfile(outputFolderPath, outputFileName);
    % 将 newData 写入 Excel 文件
    header = {'体积Volume','numL','numW','numH','L','W','H','cate'}; % 根据实际列数添加标题
    combinedData = [header; num2cell(OOdata)]; % 注意使用 num2cell 将数据矩阵转换为单元格数组
      xlswrite(outputFilePath, combinedData);
        disp(['第', num2str(orderNumber), '号装载数据已写入文件。']);

end

disp('所有订单数据处理完成。');
```

附录 三　所有订单的箱子数据

```
%% 一个订单的处理

clc;close all;clear
% 读取订单数据
orderData =
    readtable('E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\处理数据\所有订单箱子数据\1.csv');
orderData = table2array(orderData);
% 读取箱子数据
boxData = xlsread('E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\原始数据\箱子数据.xlsx');
%%
% 提取规格和数量信息
uniqueOrders = unique(orderData(:, 1));
results = [];
Quantity=[];
long=[];
wid=[];
hig=[];

% 遍历每个订单
for i = 1:length(uniqueOrders)
    order = uniqueOrders(i);
    orderRows = orderData(orderData(:, 1) == order, :);

    % 提取订单数据中规格的数量
    orderQuantities = orderRows(:, 2);
    dimensions = orderRows(:, 3:5);

    % 遍历每个规格
    for j = 1:size(orderRows, 1)
        quantity = orderQuantities(j);
        dims = dimensions(j, :);
        V=dims(1) * dims(2) * dims(3);
        length1=dims(1,1);
        width1=dims(1,2);
        high1=dims(1,3);

        for k=1:size(boxData, 1)
        % 计算最大矩形体数量
        boxDims = boxData(k, 3:5);
        V_box= prod(quantity(:,:)*boxData(k,3:5));
```

```
        maxlong=floor(boxDims(:,1)/length1);
        maxwidth=floor(boxDims(:,2)/width1);
        maxhigh=floor(boxDims(:,3)/high1);


        long=[long;maxlong];
        wid=[wid;maxwidth];
        hig=[hig;maxhigh];
        maxQuantity = ([long, wid, hig]);
%         Quantity = [Quantity;maxQuantity];
%        if maxQuantity > orderQuantities
%
%
%         end
        end



    end
end


%%
h = j; % 替换为您的需求，指示要分割的批次数量
batchSize = size(maxQuantity, 1) / h; % 计算每个批次的大小
for i = 1:h
    startIndex = (i - 1) * batchSize + 1; % 计算起始索引
    endIndex = i * batchSize; % 计算结束索引

    variableName = ['data' num2str(i)]; % 构造变量名
    dataStruct.(variableName) = maxQuantity(startIndex:endIndex, :); % 将数据赋值给相应变量
end
%%
% 初始化记录的变量
recordedData = [];
fields = fieldnames(dataStruct);
% 遍历每个 h 和 p 组合
for i = 1:length(fields)
    h = str2double(fields{i}(end)); % 提取 h 值
    datah = dataStruct.(fields{i});

    for p = 1:size(datah, 1)
        rowValues = datah(p, :);

        % 检查行数据是否全部非零
        if all(rowValues ~= 0)
```

```
            recordedData = [recordedData; h, p, rowValues];
        end
    end
end

% 根据 recordedData 中的 h 和 p 值检索相关数据
newData = [];
for i = 1:size(recordedData, 1)
    hValue = recordedData(i, 1);
    pValue = recordedData(i, 2);

    orderDataValues = orderData(hValue, 1:5);
    boxDataValues = boxData(pValue, 3:5);
    newData = [newData; orderDataValues, boxDataValues, recordedData(i, 3:end)];
end
%%

% 计算并添加新的列数据到 newData 中

l1= newData(:,9);%长的个数
l2=newData(:,10);%宽的个数
l3=newData(:,11);%高的个数
numl123=l1 .* l2 .* l3;
newData(:,12)=numl123; %一个箱子中一共能摆放的数量
for i = 1: size(newData,1)
    if newData(i,12) >= newData(i,2)
       newData(i,13) = newData(i,2); %实际用的箱子装载的个数
    else
        newData(i,13) =newData(i,12);
    end
end

newData(:, 14) = newData(:, 6) .* newData(:, 7) .* newData(:, 8) ;%箱子单个体积
newData(:, 15) = newData(:, 3) .* newData(:, 4) .* newData(:, 5) .* newData(:,13)
    ;%货物实际总体积
newData(:, 16) = ceil(newData(:, 13) .\ newData(:, 2)); % 箱子数量，向上取整，如果有余数则加 1
newData(:, 17) = newData(:, 14) .* newData(:,16) ;%使用箱子总体积
newData(:,18) = 1-(newData(:,17) .\ newData(:,15))+newData(:,16); %得分 =
    1-（实际装载的总体积/使用箱子的总体积）+箱子个数
% newData(:, end) = newData(:, 9) .* newData(:, 3) .* newData(:, 10) .* newData(:, 4) .*
    newData(:, 11) .* newData(:, 5);
%%
% 将 newData 按照第 12 列数据从大到小排序
sortedData = sortrows(newData, -12);

% 将排序后的数据保存为 needdata
needdata = sortedData;
```

```
%%
% 计算第二到第四列的和作为特征
feature_sums = sum(needdata(:, 3:5), 2);
%%
% 初始化一个列向量，用于存储类别信息
categories = zeros(size(needdata, 1), 1);

% 根据特征和计算类别
unique_sums = unique(feature_sums);
numCategories = numel(unique_sums);

for i = 1:numCategories
    matchingRows = feature_sums == unique_sums(i);
    categories(matchingRows) = i;
end



% 将类别信息添加到原数据中
data_with_categories = [needdata, categories];
OOdata=[needdata(:,15),needdata(:,9),needdata(:,10),needdata(:,11),needdata(:,3),needdata(:,4),needdata(:,5),da
%%
% %表头
% 假设您已经准备好名为 'data2_updated' 的数据矩阵

% 假设您已经准备好名为 'data2_updated' 的数据矩阵

% 创建一个假定的表头
header = {'体积Volume','numL','nunmW','numH','L','W','H','cate'}; % 根据实际列数添加标题

% 将表头和数据组合成一个矩阵
combinedData = [header; num2cell(OOdata)]; % 注意使用 num2cell 将数据矩阵转换为单元格数组


% 设置写入文件的路径
output_path = 'E:\1数学建模2023\2023国赛 () \第二次培训\数据预处理\处理数据\1号单装载.xlsx';

% % 使用 xlswrite 将整个矩阵写入 Excel 文件
% xlswrite(output_path, combinedData);

disp('数据已更新并写入新文件。');
```

附录 四　每个订单的分配

```
%% 箱子所有订单一次性处理
clc;
close all;
clear;

inputFolderPath =
    'E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\处理数据\所有订单箱子数据\';
outputFolderPath =
    'E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\处理数据\所有订单箱子装载数据\';

% 获取所有订单数据文件名
orderDataFiles = dir(fullfile(inputFolderPath, '*.csv'));

% 提取文件名
fileNames = {orderDataFiles.name};

% 提取文件名中的数字部分
fileNumbers = cellfun(@(name) sscanf(name, '%d'), fileNames);

% 对文件名按照数字部分排序
[sortedFileNumbers, sortedIndices] = sort(fileNumbers);
sortedFileNames = fileNames(sortedIndices);
sortedFileNames = sortedFileNames';
%%
 for fileIdx = 1:length(sortedFileNames)
%  for fileIdx = 1:2
    recordedData=[];
    orderRows=[];
    dataStruct = [];
   orderDataFileName = sortedFileNames{fileIdx};
   orderDataFilePath = fullfile(inputFolderPath, orderDataFileName);

   % 从文件名中提取编号部分
   [~, orderFileName, ~] = fileparts(orderDataFileName);
   orderNumber = fileNumbers(sortedIndices(fileIdx));
   orderDataFile=fullfile(orderDataFilePath);

       orderData = readtable(orderDataFile);
       orderData = table2array(orderData);
     % 读取箱子数据
       boxData =
           xlsread('E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\原始数据\箱子数据.xlsx');
% 提取规格和数量信息
uniqueOrders = unique(orderData(:, 1));
results = [];
Quantity=[];
```

```
long=[];
wid=[];
hig=[];

% 遍历每个订单
for i = 1:length(uniqueOrders)
    order = uniqueOrders(i);
    orderRows = orderData(orderData(:, 1) == order, :);

    % 提取订单数据中规格的数量
    orderQuantities = orderRows(:, 2);
    dimensions = orderRows(:, 3:5);

    % 遍历每个规格
    for j = 1:size(orderRows, 1)
        quantity = orderQuantities(j);
        dims = dimensions(j, :);
        V=dims(1) * dims(2) * dims(3);
        length1=dims(1,1);
        width1=dims(1,2);
        high1=dims(1,3);

        for k=1:size(boxData, 1)
        % 计算最大矩形体数量
        boxDims = boxData(k, 3:5);
        V_box= prod(quantity(:,:)*boxData(k,3:5));


        maxlong=floor(boxDims(:,1)/length1);
        maxwidth=floor(boxDims(:,2)/width1);
        maxhigh=floor(boxDims(:,3)/high1);


        long=[long;maxlong];
        wid=[wid;maxwidth];
        hig=[hig;maxhigh];
        maxQuantity = ([long, wid, hig]);
        end



    end
end
h = size(orderRows,1); % 替换为您的需求，指示要分割的批次数量
t= size(orderRows,1);
batchSize = size(maxQuantity, 1) / h; % 计算每个批次的大小
y=h;
```

```
for i = 1:h
    startIndex = (i - 1) * batchSize + 1; % 计算起始索引
    endIndex = i * batchSize; % 计算结束索引

    variableName = ['data' num2str(i)]; % 构造变量名
    dataStruct.(variableName) = maxQuantity(startIndex:endIndex, :); % 将数据赋值给相应变量

end

% 初始化记录的变量
recordedData = [];
fields = fieldnames(dataStruct);

% 遍历每个 h 和 p 组合
for i = 1:length(fields)
 fieldName = fields{i};
h = str2double(fieldName(5:end)); % 假设字段名形如 "data10", 提取从第 5 个字符到末尾的部分

    datah = dataStruct.(fields{i});

    for p = 1:size(datah, 1)
        rowValues = datah(p, :);

        % 检查行数据是否全部非零
        if all(rowValues ~= 0)
            recordedData = [recordedData; h, p, rowValues];
        end
    end
end

% 根据 recordedData 中的 h 和 p 值检索相关数据
newData = [];
for l = 1:size(recordedData, 1)
    hValue = recordedData(l, 1);
    pValue = recordedData(l, 2);

    orderDataValues = orderData(hValue, 1:5);
    boxDataValues = boxData(pValue, 3:5);
    newData = [newData; orderDataValues, boxDataValues, recordedData(l, 3:end)];
end
%%

% 计算并添加新的列数据到 newData 中

l1= newData(:,9);%长的个数
l2=newData(:,10);%宽的个数
l3=newData(:,11);%高的个数
```

```
numl123=l1 .* l2 .* l3;
newData(:,12)=numl123; %一个箱子中一共能摆放的数量
for i = 1: size(newData,1)
    if newData(i,12) >= newData(i,2)
       newData(i,13) = newData(i,2); %实际用的箱子装载的个数
    else
        newData(i,13) =newData(i,12);
    end
end

newData(:, 14) = newData(:, 6) .* newData(:, 7) .* newData(:, 8) ;%箱子单个体积
newData(:, 15) = newData(:, 3) .* newData(:, 4) .* newData(:, 5) .* newData(:,13)
    ;%货物实际总体积
newData(:, 16) = ceil(newData(:, 13) .\ newData(:, 2)); % 箱子数量，向上取整，如果有余数则加 1
newData(:, 17) = newData(:, 14) .* newData(:,16) ;%使用箱子总体积
newData(:,18) = 1-(newData(:,17) .\ newData(:,15))+ newData(:,16); % 得分 =
    1-（实际装载的总体积/使用箱子的总体积）+箱子个数

%%
% 将 newData 按照第 12 列数据从大到小排序
sortedData = sortrows(newData, -12);

% 将排序后的数据保存为 needdata
needdata = sortedData;

%%
% 计算第二到第四列的和作为特征
feature_sums = sum(needdata(:, 3:5), 2);
%%
% 初始化一个列向量，用于存储类别信息
categories = zeros(size(needdata, 1), 1);

% 根据特征和计算类别
unique_sums = unique(feature_sums);
numCategories = numel(unique_sums);

for i = 1:numCategories
    matchingRows = feature_sums == unique_sums(i);
    categories(matchingRows) = i;
end
% 将类别信息添加到原数据中
data_with_categories = [needdata, categories];
OOdata=[needdata(:,15),needdata(:,9),needdata(:,10),needdata(:,11),needdata(:,3),needdata(:,4),needdata(:,5),da
% recordedData=[];

    outputFileName = sprintf('第%d号装载数据.csv', orderNumber);
    outputFilePath = fullfile(outputFolderPath, outputFileName);
```

```
    % 将 newData 写入 Excel 文件
    header = {'体积Volume','numL','numW','numH','L','W','H','cate'}; % 根据实际列数添加标题
    combinedData = [header; num2cell(OOdata)]; % 注意使用 num2cell 将数据矩阵转换为单元格数组
%     xlswrite(outputFilePath, combinedData);

       disp(['第', num2str(orderNumber), '号装载数据已写入文件。']);

end

disp('所有订单数据处理完成。');
```

附录 五　得分代码

```
clc;
close all;
clear;
%这里以箱子为例
order = [7, 1];
boxData = xlsread('E:\1数学建模2023\2023国赛 () \第二次培训\数据预处理\原始数据\箱子数据.xlsx');
box = boxData(:, 3:5);
Vbox = prod(box, 2);

Res = {[3, 2]; [1, 1]; [2, 1]; [6, 3]; [6, 4]};
Res{1,1}(2,:)=[2,4];
t=[0,length(Res)];
%%
for i = 1:5
    for a = 1:size(Res{i}, 1)
        plan = Res{i}(a, 1:2);

        result = floor(order ./ plan);
        remainder = mod(order, plan);

        % 将对应的 result 加 1
        result(remainder > 0) = result(remainder > 0) + 1;
        t = max(result);

        % 在当前子元素后面添加新列，值为 t
        Res{i}(a,end+1) = t;
          while a >1
             if Res{i}(a-1,end) == 0
                Res{i}(a,end-1) = t;
                Res{i}(:,end) = [];
             end
             break
```

```
        end
    end
 end

clear i a
%%
data=struct();
% 针对每个容器创建矩阵
for idx = 1:5
    matrix = [];
    a = size(Res{idx}, 1); %第idx规格盒子的第a种方案
    b = Res{idx}(:,3) ;%第三列数值，第a种方案总共每种的最大数值

    data_name = ['data', num2str(idx)];
    data.(data_name) = matrix;

end
%%
data=[];
for i = 0:Res{1,1}(1,3)
    for j=0:Res{1,1}(2,3)
        data=[data; i, j];

    end
end
sumi= sum(data(:, 1:2), 2);
data(:, end+1) = sumi;
```

附录 六　朴素搜索递归

```
function sol=CalcSolution(p,A,B,C,Obj,quantity)
    % 单个箱子放置方案的求解函数
    % p为当前订单物体数量需求向量,(A,B,C)为当前空间的三维,quantity为当前已安排物体数量向量
    % Obj为当前订单中对当前规格箱子的简单组合体类别排列,table类型
    % 包含体积Volume,numL,numW,numH,L,W,H,cate
    q=length(Obj.cate);len=length(p);
    q = length(Obj.cate);
    len = length(p);

    if all(quantity >= p)
        sol = zeros(1, len); % A valid solution
        return
    else
        sol = []; % Initialize an empty solution
    end
```

```
    rotate=perms(1:3);sol=[];
    num=0;

    for i=1:6
        [maxLoc,maxInd]=FindMaxVol(A,B,C,Obj,rotate(i,:));
        if (maxLoc==[])
            continue
        end

        for ind=maxInd:q
            loc=Obj(ind,:);
            L=Obj.L(ind); W=Obj.W(ind); H=Obj.H(ind);
            t=[L,W,H];t=t(rotate);
            L=t(1); W=t(2); H=t(3);

            if (L<=A && W<=B && H<=C)
                loc=Obj(ind,:);
               return
            end

            clear L W H t
            quantity=quantity+[zeros(1,loc.cate-1),loc.numL*loc.numW*loc.numH,zeros(1,len-loc.cate)];%又多安排了l
            solAbove=CalcSolution(p,loc.numL * loc.L,loc.numW * loc.W,C-loc.numH *
                loc.H,Obj,quantity);
            solBeside=CalcSolution(p,loc.numL*loc.L,B-loc.numW*loc.W,C,Obj,quantity);
            solAhead=CalcSolution(p,A-loc.numL*loc.L,B,C,Obj,quantity);

            for a=1:length(solAbove)
                for b=1:length(solBeside)
                    for c=1:length(solAhead)
                        num=num+1;
                       sol(num,:)=quantity+solAbove(a,:)+solBeside(b,:)+solAhead(c,:);
                    end
                end
            end

combine = [solAbove; solBeside; solAhead];
combine = combine + quantity;
sol = [sol; combine]; % 将不同分支的解合并

        end
        clear a b c

sol = unique(sol, 'rows'); % 去除重复的解
    end
    clear i
end
```

附录 七　遗传算法

```
%% 清空环境
clc
clear

%% 遗传算法参数
maxgen=40;                      %进化代数
sizepop=100;                    %种群规模
pcross=[0.7];                   %交叉概率
pmutation=[0.01];               %变异概率
lenchrom=[1 1];                 %变量字串长度
bound=[-2 2;-2 2];              %变量范围

%% 个体初始化
individuals=struct('fitness',zeros(1,sizepop), 'chrom',[]); %种群结构体
avgfitness=[];                                          %种群平均适应度
bestfitness=[];                                         %种群最佳适应度
bestchrom=[];                                           %适应度最好染色体
% 初始化种群
for i=1:sizepop
    individuals.chrom(i,:)=Code(lenchrom,bound); %随机产生个体
    x=individuals.chrom(i,:);
    individuals.fitness(i)=fun(x);              %个体适应度
end

%找最好的染色体
[bestfitness bestindex]=max(individuals.fitness);
bestchrom=individuals.chrom(bestindex,:); %最好的染色体
avgfitness=sum(individuals.fitness)/sizepop; %染色体的平均适应度
% 记录每一代进化中最好的适应度和平均适应度
trace=[];
chrom = [];

%% 进化开始
for i=1:maxgen

    % 选择操作
    individuals=Select(individuals,sizepop);
    avgfitness=sum(individuals.fitness)/sizepop;
    % 交叉操作
    individuals.chrom=Cross(pcross,lenchrom,individuals.chrom,sizepop,bound);
    % 变异操作
```

```
    individuals.chrom=Mutation(pmutation,lenchrom,individuals.chrom,sizepop,[i maxgen],bound);

   % 计算适应度
   for j=1:sizepop
       x=individuals.chrom(j,:);
       individuals.fitness(j)=fun(x);
   end

 %找到最小和最大适应度的染色体及它们在种群中的位置
   [newbestfitness,newbestindex]=max(individuals.fitness);
   [worestfitness,worestindex]=min(individuals.fitness);
   % 代替上一次进化中最好的染色体
   if bestfitness<newbestfitness
       bestfitness=newbestfitness;
       bestchrom=individuals.chrom(newbestindex,:);
   end
   individuals.chrom(worestindex,:)=bestchrom;
   individuals.fitness(worestindex)=bestfitness;

   avgfitness=sum(individuals.fitness)/sizepop;
   chrom = [chrom;bestchrom];
   trace=[trace;avgfitness bestfitness]; %记录每一代进化中最好的适应度和平均适应度
end
%进化结束
```

附录 八　组合体递归

```
function [loc,ind]=FindMaxVol(A,B,C,Obj,rotate)
   len=length(Obj.cate);
   loc=[];
   for ind=1:len
       L=Obj.L(ind); W=Obj.W(ind); H=Obj.H(ind);
       t=[L,W,H];t=t(rotate);
       L=t(1); W=t(2); H=t(3);
       if (L<=A && W<=B && H<=C)
           loc=Obj(ind,:);
           return
       end
   end
end
```

附录 九　RA-BF 模型主程序代码

```
clc;close all;clear
% 读取数据
data = readtable('E:\1数学建模2023\2023国赛（）\第二次培训\解决算法\1号单装载.xlsx');
box = xlsread('E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\原始数据\箱子数据.xlsx');
orderdata=xlsread('E:\1数学建模2023\2023国赛（）\第二次培训\数据预处理\原始数据\订单数据.xlsx');
%%
% 输入检索订单号
n = 1; % 你可以将 n 设置为你想要检索的订单号
% 在订单数据中检索订单号为 n 的行
indices = orderdata(:, 1) == n;
order_items = orderdata(indices, :);

% 从订单数据中获取最后一列数据
p = order_items(:, end)';
A  =box(:,3);
B = box (:,4);
C=  box(:,5);
quantity = zeros(1, length(p)); % 初始物体数量向量
Obj=data;

%% 求解最优方案
num=1;%订单总数


for i=1:num
    for cate=1:5
        Res{i,cate}=CalcSolution(p,A(cate),B(cate),C(cate),Obj(:,:),quantity);
    end
end
clear i

% Res=cell(num,5);%装箱方案计算


%%
function res=Filter(p,origin)
    num=size(origin,1);
    count=0;
    for i=1:num
        tmp=origin(i,:);
        check=find(tmp-p)<0;%查找是否有小于0的值，全为0表示满足条件
        if (any(check) ==0)%检查
          count=count+1;
          res(count,:)=tmp;
        end
    end
```

```
end
%%
function sol = CalcSolution(p, A, B, C, Obj, quantity)
    % 单个箱子放置方案的求解函数
    % p为当前订单物体数量需求向量,(A,B,C)为当前空间的三维,quantity为当前已安排物体数量向量
    % Obj为当前订单中对当前规格箱子的简单组合体类别排列,table类型
    % 包含体积Volume,numL,numW,numH,L,W,H,cate
    q = length(Obj.cate);
    len = length(p);

    % 检查已安排数量是否满足需求
    if (any((quantity - p) < 0) == 0)
        sol = zeros(1, len);
        return;
    end
    sol = zeros(1, len);
    rotate = perms(1:3);
    num = 0;

    % 遍历不同的旋转排列方式
    for i = 1:6
        [maxLoc, maxInd] = FindMaxVol(A, B, C, Obj, rotate(i, :));

        % 如果没有适合的放置位置，继续下一个旋转排列方式
        if (maxLoc == [])
            continue;
        else
            % 遍历从最大体积物体索引开始的所有物体
            for ind = maxInd:q
                L = Obj.L(ind); W = Obj.W(ind); H = Obj.H(ind);
                t = [L, W, H];
                t = t(rotate);
                L = t(1); W = t(2); H = t(3);

                % 判断是否有合适的放置位置
                if (L <= A && W <= B && H <= C)
                    loc = Obj(ind, :);
                    return;
                end

                % 更新已安排物体数量向量
                quantity = quantity + [zeros(1, loc.cate - 1), loc.numL * loc.numW * loc.numH,
                    zeros(1, len - loc.cate)];

                % 递归调用不同放置情况
                solAbove = CalcSolution(p, loc.numL * loc.L, loc.numW * loc.W, C - loc.numH *
                    loc.H, Obj, quantity);
```

```
            solBeside = CalcSolution(p, loc.numL * loc.L, B - loc.numW * loc.W, C, Obj,
                quantity);
            solAhead = CalcSolution(p, A - loc.numL * loc.L, B, C, Obj, quantity);

            % 合并不同分支的解
            for a = 1:length(solAbove)
                for b = 1:length(solBeside)
                    for c = 1:length(solAhead)
                        num = num + 1;
                        sol(num, :) = quantity + solAbove(a, :) + solBeside(b, :) +
                            solAhead(c, :);
                    end
                end
            end
            combine = [solAbove; solBeside; solAhead];
            combine = combine + quantity;
            sol = [sol; combine]; % 将不同分支的解合并
        end
    end
end
end
%%
function [loc,ind]=FindMaxVol(A,B,C,Obj,rotate)
    len=length(Obj.cate);
    loc=[];
    for ind=1:len
        L=Obj.L(ind); W=Obj.W(ind); H=Obj.H(ind);
        t=[L,W,H];t=t(rotate);
        L=t(1); W=t(2); H=t(3);
        if (L<=A && W<=B && H<=C)
            loc=Obj(ind,:);
            return
        end
    end
end
```

附录 十　装配文件

由于数据量过于大,提供百度云链接方便查看链接:$https://pan.baidu.com/s/1EDuHq7_RUz1fa421n$ 提取码: 421n –来自百度网盘超级会员 V4 的分享