参赛编号:YRDMCM202301192
选题: A　参赛赛道:本科生

2023 年第三届长三角高校数学建模竞赛

题　　目　基于多目标规划的三维快递装箱问题

摘　　要：

本文通过对耗材进行坐标系建立，将物品能否被耗材装下的问题转化为坐标与耗材长、宽、高之间关系的问题，建立了物品装载模型，在此基础上对耗材尺寸进行优化，并进一步考虑耗材具有柔性属性对模型进行调整。

针对问题一，问题一要求分别用箱子、袋子以及两种耗材共同使用的情况下对附件中的订单物品进行装载。要求在满足耗材使用数量尽量少的前提下尽量减小耗材总体积。本文以耗材的端点为原点建立坐标系，标记物体距离原点最远的棱角的坐标，当用箱子装载物品时，要求该坐标小于耗材的对应尺寸，当用袋子装时，要求该坐标满足提示中的两条不等式。另外，物体被装载时在耗材内部不应该存在重合的情况。最终，以耗材使用数量尽量少为第一目标，耗材总体积尽量小为第二目标，上述条件为约束建立**物品装载模型**。在求解模型时，创建一个未被装载物品列表，每次选择耗材装载物品后都更新该列表，当更新后的物品列表为空列表时，表示所有物品都被装载完成，输出耗材数量和总体积即可。得到仅用箱子装载的结果为：箱子耗材使用总数为 8173，总体积为 74064326000。仅用袋子装载的结果为：袋子耗材总数为 16442，总体积为 2610000000。得到既能用箱子装也能用袋子装的结果为：耗材使用总数为 6309，总体积为 13131824000。

针对问题二，问题二要求改变耗材尺寸，本问中模型仍要达成的目标不变，因此，本文以耗材数量尽量少为第一目标，耗材总体积尽量小为第二目标，在问题一物品装载模型约束条件的基础上加入增量约束作为约束条件，建立**耗材尺寸优化模型**。求解模型时，首先对问题一中装载成功的每个订单所含物体的总体积用 **K-Means 法**进行聚类，根据不同的耗材聚成不同数量的类别，再在每一类中选取订单最大体积，该类耗材的体积应大于等于该类订单的最大体积。通过循环搜索得到能够满足容积以及模型约束条件的耗材。得到最终仅用箱子装载的 5 种耗材尺寸为：[417,196,300]，[647,215,281]，[702,250,468]，[750,400,690]，[909,569,724]。仅用箱子的使用数量为 4198，总体积为 126988550054。仅用袋子装载的 4 种耗材尺寸为：[713,492,4]，[919,487,9]，[1424,1074,16]，[1614,1274,19]。袋子使用数量为 8169，总体积为 42337650042。

针对问题三，只需要赋予长、宽、高 5%的伸展区域即可，建立**柔性条件下的装载模型**。在模型求解时，令耗材的长、宽、高都伸展 5%，对物品进行装载，标记物品距离原点最远端点的坐标，将该坐标与耗材的长、宽、高比较，若比耗材的尺寸大，则选取对应坐标为耗材的长或宽或高，否则就选择原本耗材的尺寸。如此求解得到仅用箱子使用总数为 7608，总体积为 388069000000；仅用袋子使用总数为 16278，总体积为 3210000000；自由用材耗材使用总数为 10523，总体积为 301000000000。

关键词：物品装载模型　耗材尺寸优化模型　K-Means 法　柔性条件下的装载模型

目录

正文

一、 问题重述

1.1 问题背景

近年来，中国快递行业飞速发展，2022 年一年内的包裹数量已超 1000 亿件，中国凭借强大、先进、完善的快递物流体系俨然成为世界快递大国。由于极大的包裹基数，为取得更高的经济效益，对每个包裹耗材提出了更高的要求，即使略微降低包裹耗材的成本，也能够获得极大的经济收益。附件中给出了订单数据和耗材数据，基于这些数据，探索如何装箱才能够使得耗材数量尽量少，耗材利用率尽量高，从而提高经济效益。

1.2 三个任务

任务一：任务完成者需要根据附件给出的订单数据，对每个订单采用箱装或袋装的方法进行处理，建立合理的装箱模型，该模型需满足两个要求，一是使耗材数尽量少，二是在耗材数尽量少的前提下使得耗材总体积尽量小。最后，给出具体的装箱方案，包括耗材的数量和耗材总体积。

任务二：任务完成者需要根据附件数据在耗材种数不变的前提下优化耗材尺寸，要求优化后的方案装载物品的耗材数量能够尽量减少，在此前提下，耗材的总体积也应尽量小，且不超过原方案的总体积。给出具体的耗材尺寸优化方案，以及优化后每种耗材的具体尺寸、使用数量和耗材总体积。

任务三：任务完成者需要考虑货物和耗材具备能够被轻微挤压的柔性属性，重新完成前面两个任务，要求耗材伸展时的长、宽、高都不能超过原尺寸的 5%。

二、 问题分析

2.1 问题一的分析

问题一要求用袋子或箱子对附件中的订单进行装载，因此，耗材的种类和形状会对不同物品的装载方法进行约束。根据提示，物品的长、宽、高并不限定，故对于每一种物品，长、宽、高的组合共有$3\times2=6$种。考虑通过坐标方法对每一个物品的不同长、宽、高组合建立物品装载的约束条件，包括所有装载物品的总长、总宽和总高度不能超过耗材的长、宽、高，耗材数量应不超过订单物品数量，同一耗材内的物品在空间上不能有重合的部分等。然后，以耗材数量尽可能少为第一目标，耗材总体积尽量小为第二目标建立物品装载的多目标规划模型。最后，采用主次目标法对模型进行求解，分别得到只用袋装，只用箱装和两种耗材同时使用的具体方案。

2.2 问题二的分析

问题二仅对耗材的尺寸进行优化，耗材种类仍保持 9 种不变，故只需要在第一问物品装载模型的基础上对耗材的长、宽、高赋予一个增量即可。根据附件数据，物品和耗材的长、宽、高都为整数，因此本文考虑赋予耗材的增量也为整数。对袋子而言，由于 4 种袋子耗材的高度都为 1，故袋子高度的增量一定为正值，而对于袋子的长、宽以及箱子而言，增量可正可负。以耗材数量尽量少为第一目标，耗材总体积尽量小为第二目标，约束条件为在问题一物品装载模型约束条件的基础上加入增量这一变量，从而建立耗材尺寸优化模型。求解模型即可得到具体的耗材尺寸优化方案以及优化后每种耗材的

使用数量和耗材总体积，客观上耗材的使用数量和耗材总体积应不超过问题一模型求解所得到的结果。

2.3 问题三的分析

问题三与问题一、二的不同之处在于，加入了耗材的柔性条件，更加贴近实际情况。在物品允许被轻微挤压的情况下，耗材伸展时的长、宽、高尺寸的变化不能超过原尺寸的 5%。问题三要求在此条件上重新完成前两问，只需要在前两问已经建立的模型中将耗材的长、宽、高从定值改为变量，并对其进行 5%条件的约束。模型求解的方法与前两问的求解方法类似，仅增加了对长、宽、高变量的遍历搜索，最终得到前两问在耗材具有柔性条件下的求解结果。

三、 模型假设

1.**假设货物装箱后结构稳定，**在求出装载方案后，物品不会因为耗材中所存在的空隙而发生翻倒等情况。

2.**假设耗材厚度可以忽略，**否则在问题求解过程中耗材的体积不是实际可容纳物品的体积，还需要排除厚度的影响。

3.**假设耗材在伸展过程中不会发生破损，**否则模型需要考虑由破损产生的耗材使用数量，从而对模型的建立造成阻碍。

四、 符号说明

符号	说明
C_{im}	i订单使用第m种耗材的数量
V_m	第m种耗材的体积
E_{ijn}	i订单j种类第n个物品不同长、宽、高组合的变量，取值为 1~6
N_{ij}	i订单所含j种类物品物品的数量
L_{ij}	i订单j种类物品的长
W_{ij}	i订单j种类物品的宽
H_{ij}	i订单j种类物品的高

注：未列出或重复的符号以文中出现处为准

五、 模型准备

题目要求用箱子和袋子对物品进行装载，而附件订单中所包含的某些物品无论如何不能被袋子或箱子装下，因此，需要对这些无法进行装载的订单进行清洗，以免影响后续计算结果同时提高计算速度。

●箱装模型下的数据清洗

在仅用箱子对物品进行装载时，将订单中的所有物品尺寸与最大箱子的尺寸进行比较，判断该物品能够被装下，根据附件数据，普通 5 号自营纸箱为尺寸最大的纸箱，若

该物品不能被 5 号纸箱装下，则该物品不能被 1~5 号纸箱装下，应在计算时将其排除。根据 5 号纸箱的尺寸：长 300，宽 200，高 170，本文给出具体的数据清洗办法如下：

Step1：剔除附件中长、宽、高大于 300 的物品。

Step2：在剩余物品中进行如下判断：若物品的一条边在 $[200,300]$ 之间，判断是否存在另一条边在 $[200,300]$ 之间，若存在，则删除该行，否则进行 step3。

Step3：若物品的一条边在 $[170,200]$ 之间，判断是否存在另一条边在 $[170,200]$ 之间，若存在，则删除该行。

经过上述处理办法，总计剔除了 935 个订单、4415 个物品，剩余 4198 个订单、21422 个物品参与计算。

●袋装模型下的数据清洗

根据题给的两条不等式：袋子长与高的和大于等于物品长与高的和且袋子宽与高的和大于等于物品宽与高的和。由于附件中给的物品数据长大于宽大于高，因此，将物品的长和高相加，宽和高相加，与普通 4 号袋的尺寸进行比较，若物品的长加高大于 451 或物品的宽加高大于 421，则剔除这些行。

经过上述处理办法，总计剔除了 302 个订单、1093 个物品，剩余 4831 个订单、24744 个物品参与计算。

●自由用材模型下的数据清洗

在在这一部分，需要剔除及不满足箱装条件也不满足袋装条件的订单和物品，总计剔除了 292 个订单、748 个物品，剩余 4841 个订单、25089 个物品

六、 模型的建立与求解

6.1 问题一模型的建立与求解

6.1.1 问题一的分析

问题一要求用袋子或箱子对附件中的订单进行装载，因此，耗材的种类和形状会对不同物品的装载方法进行约束。本文通过坐标方法建立物品装载的约束条件，以耗材数量尽可能少为第一目标，耗材总体积尽量小为第二目标建立物品装载的多目标规划模型。最后，采用主次目标法进行求解。

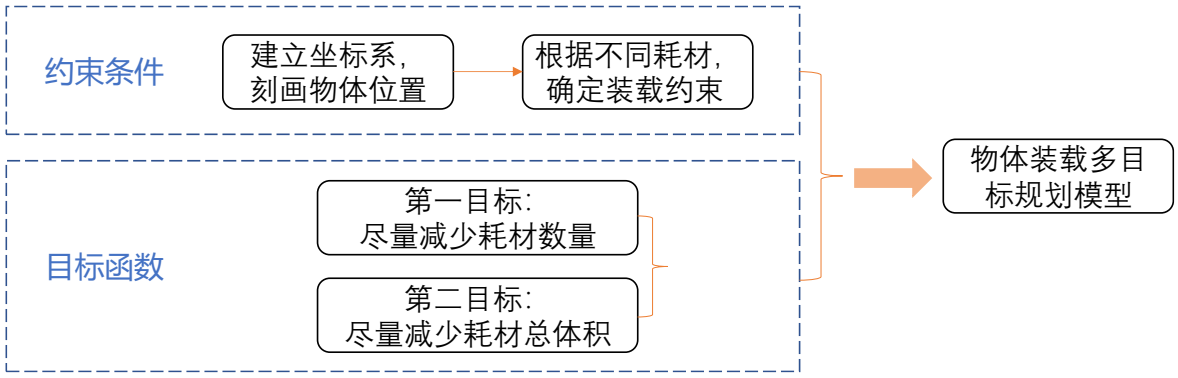

图 1：问题一模型结构分析图

6.1.2 物品装载模型

由于箱子的尺寸固定，而袋子具有伸展性，因此，物品装载模型需要根据耗材分为两种情况进行讨论，即为用箱子装载物品和用袋子装载物品，分别给出两种情况下耗材能够装载物理所必须满足的条件。

(一)箱装模型

这一部分，本文仅使用箱子种类的耗材对订单进行装载，根据清洗后的数据，附件中能够被箱子装下的订单为4198，物品数量为21422。根据题目要求，在尽量减少耗材使用数量的前提下使得耗材总体积尽量小。因此有如下目标函数。

●目标函数

第一目标：在完成订单中的物品装载时，应尽量减少耗材的使用数量。假设C_{im}表示i订单使用第$m(m=5,6,\cdots,9)$种耗材的数量，则第一目标的数学表达如下：

$$\min \sum_{i=1}^{4198} \sum_{m=5}^{9} C_{im} \tag{1}$$

第二目标：在满足第一目标的前提下，即保持使用耗材总数不变的前提下尽量减少耗材使用的总体积。假设V_m表示第m种耗材的体积，则第二目标的数学表达为：

$$\min \sum_{i=1}^{4198} \sum_{m=5}^{9} C_{im} \cdot V_m \tag{2}$$

●决策变量

在本题中，假设箱子类型的耗材尺寸固定，因此在建立箱装模型时，需要满足装载物品的总长、总宽和总高不能超过耗材的长、宽、高。而根据题意，物品的长、宽、高并不固定，经过排列组合共有$3\times2=6$种情况，需要分别把这 6 种情况分别表达出来。下面设变量$E_{ijn}=1,2,\cdots,6$分别对应i订单j种类第n个物品 6 种不同的长、宽、高组合情况，即：

$$1\leqslant E_{ijn}\leqslant 6,\quad E_{ijn}\in Z^{+} \tag{3}$$

$E_{ijn}=1,2,\cdots,6$对应如下六种情况：

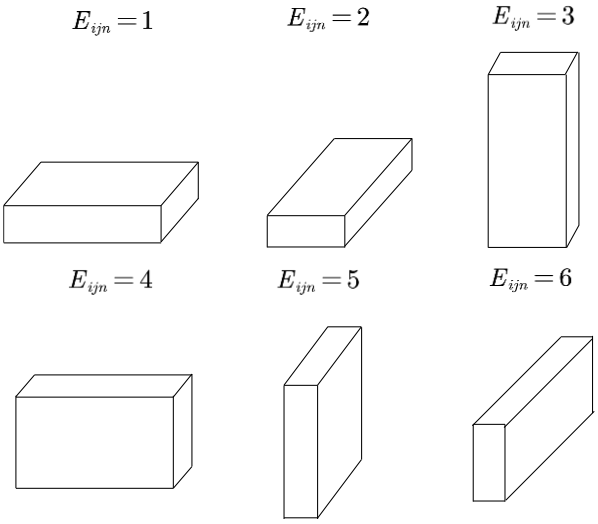

图 2：长宽高组合示意图

本题的决策变量为E_{ijn}。

●约束条件

而为了定位不同情况下物品在耗材中的位置，本文通过建立坐标系，确定物品坐标的方法进行位置刻画，考虑以耗材某个棱角为原点建立坐标系，如下图所示：

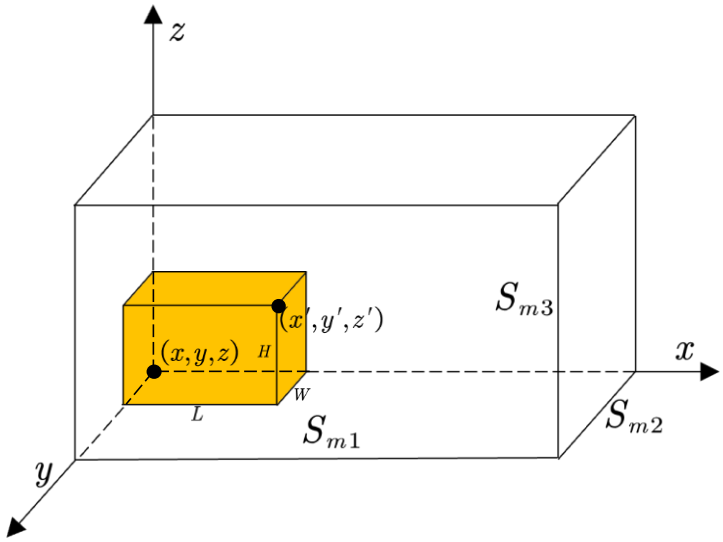

图 3：物品位置示意图

以箱型耗材的其中一个棱角作为原点，相邻的三条棱作为坐标轴的三个方向建立坐标系。以上图中物品为例，假设长、宽、高分别为L、W、H，标记最靠近原点的棱角坐标为(x,y,z)，离原点最远的棱角坐标为(x',y',z')，则两点之间的量化关系为：

$$x'=x+L,y'=y+W,z'=z+H \tag{4}$$

进一步，根据附件数据特点，假设i订单j种类第n个物品所装载物品距离原点最近棱角的三个坐标分别为$LL_{ijn}(x)$、$LL_{ijn}(y)$、$LL_{ijn}(z)$，则物品的最远端点的三个坐标分别为$LL'_{ijn}(x)$、$LL'_{ijn}(y)$、$LL'_{ijn}(z)$，假设i订单j种类物品的长、宽、高分别为L_{ij}、W_{ij}、H_{ij}，第$m(m=5,6,\cdots,9)$种耗材的长、宽、高分别为S_{1m}、S_{2m}、S_{3m}，则对应$E_i=1,2,\cdots,6$情况下两个点之间的坐标关系分别为：

$$\begin{cases} LL'_{ijn}(x)=LL_{ijn}(x)+L_{ij},LL'_{ijn}(y)=LL_{ijn}(y)+W_{ij},LL'_{ijn}(z)=LL_{ijn}(z)+H_{ij},E_{ijn}=1 \\ LL'_{ijn}(x)=LL_{ijn}(x)+W_{ij},LL'_{ijn}(y)=LL_{ijn}(y)+L_{ij},LL'_{ijn}(z)=LL_{ijn}(z)+H_{ij},E_{ijn}=2 \\ \vdots \\ LL'_{ijn}(x)=LL_{ijn}(x)+H_{ij},LL'_{ijn}(y)=LL_{ijn}(y)+W_{ij},LL'_{ijn}(z)=LL_{ijn}(z)+L_{ij},E_{ijn}=5 \\ LL'_{ijn}(x)=LL_{ijn}(x)+H_{ij},LL'_{ijn}(y)=LL_{ijn}(y)+L_{ij},LL'_{ijn}(z)=LL_{ijn}(z)+W_{ij},E_{ijn}=6 \end{cases} \tag{5}$$

物品被装载在耗材内仅需物品距原点最远端的点坐标小于耗材的对应长、宽、高即可，因此，有如下约束：

$$LL'_{ijn}(x)\leqslant S_{1m},LL'_{ijn}(y)\leqslant S_{2m},LL'_{ijn}(z)\leqslant S_{3m} \tag{6}$$

另外，模型应满足对于第i个订单，使用的总耗材数量应不超过订单所包含的物品总数量，假设J_i表示i订单所含物品的种类数，N_{ij}表示i订单所含j种类物品物品的数量，则有：

$$\sum_{m=5}^{9}C_{im}\leqslant\sum_{j=1}^{J_i}N_{ij} \tag{7}$$

同时，在装箱过程中，箱中物品不应该在空间上存在重合的部分，本文将三维的重合问题通过投影转化为二维平面的重合问题，空间物品的重合情况如下图所示：

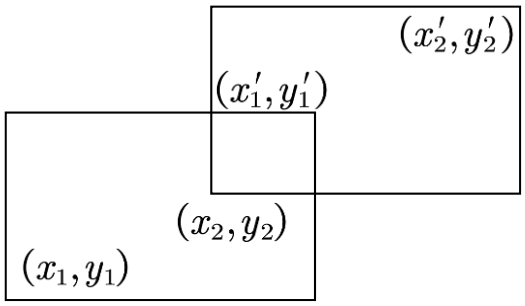

图 4：二维平面重合示意图

图中两个长方形代表空间中物品投影的两个平面，若这两个平面有交集，则空间上这两个物品有重合的部分，不符合客观规律。要使空间上物品不重合，对每个方向的投影而言，只需满足如下两个不等式：

$$\begin{cases} \max(x_1, x_2) \geqslant \min(x_1', x_2') \\ \max(y_1, y_2) \geqslant \min(y_1', y_2') \end{cases} \tag{8}$$

假设 0-1 变量$G(x,y)$表示两个平面是否相交，$G(x,y)$的值为 0 时，代表两个平面相交，$G(x,y)$的值为 1 时，代表两个平面不相交，综合上式，有：

$$G(x,y) = \begin{cases} 1, \ \max(x_1, x_2) \geqslant \min(x_1', x_2') \ \ and \ \ \max(y_1, y_2) \geqslant \min(y_1', y_2') \\ 0, else \end{cases} \tag{9}$$

将一般情况特殊化，针对订单中的具体物品，考虑任意两个物品间的重合关系，有：

$$G(LL_{ijn}(x), LL_{ijn}(y)) = \begin{cases} 1, \ \begin{cases} \max(LL_{ij'n'}(x), LL_{ij''n''}(x)) \geqslant \min(LL'_{ij'n'}(x), LL'_{ij''n''}(x)) \\ and \\ \max(LL_{ij'n'}(y), LL_{ij''n''}(y)) \geqslant \min(LL'_{ij'n'}(y), LL'_{ij''n''}(y)) \end{cases} \\ 0, else \end{cases} \tag{10}$$

空间物品不重合应对应 3 个投影方向下的二维平面都不重合，故需要满足如下表达式：

$$G(LL_{ijn}(x), LL_{ijn}(y)) \cdot G(LL_{ijn}(y), LL_{ijn}(z)) \cdot G(LL_{ijn}(z), LL_{ijn}(x)) = 1 \tag{11}$$

●箱装模型建立

综合上述分析，将第一目标作为第二目标完成的前提，建立主次目标下箱装的多目标规划模型：

$$\min \sum_{i=1}^{4198} \sum_{m=5}^{9} C_{im} \cdot V_m$$

$$\begin{cases} 1 \leqslant E_{ijn} \leqslant 6, \ E_{ijn} \in Z^+ \\ V_m = S_{1m} \cdot S_{2m} \cdot S_{3m} \\ \begin{cases} LL'_{ijn}(x) = LL_{ijn}(x) + L_{ij}, LL'_{ijn}(y) = LL_{ijn}(y) + W_{ij}, LL'_{ijn}(z) = LL_{ijn}(z) + H_{ij}, E_{ijn} = 1 \\ LL'_{ijn}(x) = LL_{ijn}(x) + W_{ij}, LL'_{ijn}(y) = LL_{ijn}(y) + L_{ij}, LL'_{ijn}(z) = LL_{ijn}(z) + H_{ij}, E_{ijn} = 2 \\ \vdots \\ LL'_{ijn}(x) = LL_{ijn}(x) + H_{ij}, LL'_{ijn}(y) = LL_{ijn}(y) + W_{ij}, LL'_{ijn}(z) = LL_{ijn}(z) + L_{ij}, E_{ijn} = 5 \\ LL'_{ijn}(x) = LL_{ijn}(x) + H_{ij}, LL'_{ijn}(y) = LL_{ijn}(y) + L_{ij}, LL'_{ijn}(z) = LL_{ijn}(z) + W_{ij}, E_{ijn} = 6 \end{cases} \\ LL'_{ijn}(x) \leqslant S_{1m}, LL'_{ijn}(y) \leqslant S_{2m}, LL'_{ijn}(z) \leqslant S_{3m} \\ \sum_{m=5}^{9} C_{im} \leqslant \sum_{j=1}^{J_i} N_{ij} \\ G(LL_{ijn}(x), LL_{ijn}(y)) \cdot G(LL_{ijn}(y), LL_{ijn}(z)) \cdot G(LL_{ijn}(z), LL_{ijn}(x)) = 1 \\ G(LL_{ijn}(x), LL_{ijn}(y)) = \begin{cases} 1, \ \begin{cases} \max(LL_{ij'n'}(x), LL_{ij''n''}(x)) \geqslant \min(LL'_{ij'n'}(x), LL'_{ij''n''}(x)) \\ and \\ \max(LL_{ij'n'}(y), LL_{ij''n''}(y)) \geqslant \min(LL'_{ij'n'}(y), LL'_{ij''n''}(y)) \end{cases} \\ 0, else \end{cases} \\ \min \sum_{i=1}^{4198} \sum_{m=5}^{9} C_{im} \\ LL_{ijn}(x) \geqslant 0, LL_{ijn}(y) \geqslant 0, LL_{ijn}(z) \geqslant 0 \end{cases} \tag{12}$$

(二)袋装模型

这一部分，本文仅使用袋子种类的耗材对订单进行装载，根据清洗后的数据，附件中能够被袋子装下的订单为 4831，物品数量为 24744。袋装模型与箱装模型的相同之处在于，他们所需要达到的目标相同，而由于耗材形态的变化，对应的约束条件也会发生相应的改变。

●目标函数

第一目标：假设C_{im}表示i订单使用第$m(m=1,2,\cdots,4)$种耗材的数量，则尽量减少耗材的使用数量的数学表达如下：

$$\min \sum_{i=1}^{4831} \sum_{m=1}^{4} C_{im} \tag{13}$$

第二目标：在满足第一目标的前提下，尽量减少耗材使用的总体积。假设V_m表示第m种耗材的体积，则第二目标的数学表达为：

$$\min \sum_{i=1}^{4831} \sum_{m=1}^{4} C_{im} \cdot V_m \tag{14}$$

●决策变量

与箱装模型相同，本题的决策变量仍为代表i订单j种类第n个物品不同长、宽、高组合的变量$E_{ijn}, E_{ijn}=1,2,\cdots,6$。

●约束条件

首先，对于第i个订单，使用的总耗材数量应不超过订单所包含的物品总数量。假设J_i表示i订单所含物品的种类数，N_{ij}表示i订单所含j种类物品物品的数量，则有：

$$\sum_{m=1}^{4} C_{im} \leq \sum_{j=1}^{J_i} N_{ij} \tag{15}$$

其次，根据题意，当满足袋子长与高的和大于等于物品长与高的和且袋子宽与高的和大于等于物品宽与高的和时，该物品能够被袋子装下。假设m种类袋子的长、宽、高分别为S_{1m}、S_{2m}、S_{3m}，物品距离原点物品的最远端点的三个坐标分别为$LL'_{ijn}(x)$、$LL'_{ijn}(y)$、$LL'_{ijn}(z)$，假设(i,j,n)表示i订单所含j种类第n个物品，U_{ikq}表示i订单使用的第q个k种类耗材所容纳物品的集合，则需要满足的两条关系式的数学表达如下：

$$\begin{cases} S_{1m}+S_{3m} \geq \max\limits_{(i,j,n)\in U_{ikq}} \{LL'_{ijn}(x)\} + \max\limits_{(i,i,n)\in U_{ikq}} \{LL'_{ijn}(z)\} \\ S_{2m}+S_{3m} \geq \max\limits_{(i,j,n)\in U_{ikq}} \{LL'_{ijn}(y)\} + \max\limits_{(i,i,n)\in U_{ikq}} \{LL'_{ijn}(z)\} \end{cases} \tag{16}$$

最后，同一耗材内的不重叠问题与装箱模型中的讨论相同，本文不再赘述，所确定的约束条件与式(10)、(11)相同。

●袋装模型建立

综合上述分析，只需将第一目标作为前提条件放入约束条件即可建立主次目标下的袋装多目标规划模型。

$$\min \sum_{i=1}^{4831} \sum_{m=1}^{4} C_{im} \cdot V_m$$

$$\begin{cases} 1 \leq E_{ijn} \leq 6,\ E_{ijn} \in Z^+ \\ \begin{cases} LL'_{ijn}(x)=LL_{ijn}(x)+L_{ij}, LL'_{ijn}(y)=LL_{ijn}(y)+W_{ij}, LL'_{ijn}(z)=LL_{ijn}(z)+H_{ij}, E_{ijn}=1 \\ LL'_{ijn}(x)=LL_{ijn}(x)+W_{ij}, LL'_{ijn}(y)=LL_{ijn}(y)+L_{ij}, LL'_{ijn}(z)=LL_{ijn}(z)+H_{ij}, E_{ijn}=2 \\ \vdots \\ LL'_{ijn}(x)=LL_{ijn}(x)+H_{ij}, LL'_{ijn}(y)=LL_{ijn}(y)+W_{ij}, LL'_{ijn}(z)=LL_{ijn}(z)+L_{ij}, E_{ijn}=5 \\ LL'_{ijn}(x)=LL_{ijn}(x)+H_{ij}, LL'_{ijn}(y)=LL_{ijn}(y)+L_{ij}, LL'_{ijn}(z)=LL_{ijn}(z)+W_{ij}, E_{ijn}=6 \end{cases} \\ V_m = S_{1m} \cdot S_{2m} \cdot S_{3m} \\ S_{1m}+S_{3m} \geq \max\limits_{(i,j,n)\in U_{ikq}} \{LL'_{ijn}(x)\} + \max\limits_{(i,i,n)\in U_{ikq}} \{LL'_{ijn}(z)\} \\ S_{2m}+S_{3m} \geq \max\limits_{(i,j,n)\in U_{ikq}} \{LL'_{ijn}(y)\} + \max\limits_{(i,i,n)\in U_{ikq}} \{LL'_{ijn}(z)\} \\ \sum\limits_{m=1}^{4} C_{im} \leq \sum\limits_{j=1}^{J_i} N_{ij} \\ G(LL_{ijn}(x),LL_{ijn}(y)) \cdot G(LL_{ijn}(y),LL_{ijn}(z)) \cdot G(LL_{ijn}(z),LL_{ijn}(x)) = 1 \\ G(LL_{ijn}(x),LL_{ijn}(y)) = \begin{cases} 1, & \begin{cases} \max(LL_{ij'n'}(x),LL_{ij''n''}(x)) \geq \min(LL'_{ij'n'}(x),LL'_{ij''n''}(x)) \\ and \\ \max(LL_{ij'n'}(y),LL_{ij''n''}(y)) \geq \min(LL'_{ij'n'}(y),LL'_{ij''n''}(y)) \end{cases} \\ 0, else \end{cases} \\ \min \sum\limits_{i=1}^{4831} \sum\limits_{m=1}^{4} C_{im} \\ LL_{ijn}(x) \geq 0, LL_{ijn}(y) \geq 0, LL_{ijn}(z) \geq 0 \end{cases} \tag{17}$$

(三)自由用材装载模型

这一部分，对订单中物品的装载可以同时使用两种耗材，因此，只需要将箱装模型和袋装模型结合起来，根据不同的耗材确定不同的约束条件即可。

●目标函数

假设m取值 1~9 表示附件中的 9 种不同耗材，m取值 1~4 表示耗材为袋子，m取值 5~9 表示耗材为箱子。则耗材使用数量尽量少目标的数学表达为：

$$\min \sum_{i=1}^{4841} \sum_{m=1}^{9} C_{im} \tag{18}$$

在上述目标达成前提下，耗材使用总体积尽量小的目标为：

$$\min \sum_{i=1}^{4841} \sum_{m=1}^{9} C_{im} \cdot V_m \tag{19}$$

●决策变量

决策变量仍为代表i订单j种类第n个物品不同长、宽、高组合的变量E_{ijn}，E_{ijn}为 1~6 之间的整数。

●约束条件

若$m \in [1,4],\ m \in Z^+$，则约束条件与袋装模型的约束条件相同，为式(5)、(10)、(11)、(15)、(16)，若$m \in [5,9],\ m \in Z^+$，则约束条件与箱装模型的约束条件相同，为式(5)、(6)、(7)、(10)、(11)。

●自由用材装载模型建立

综合上述分析，建立主次目标下自由用材装载的多目标规划模型：

$$\min \sum_{i=1}^{4841} \sum_{m=1}^{9} C_{im} \cdot V_m$$

$$\begin{cases}
1 \leqslant E_{ijn} \leqslant 6,\ E_{ijn} \in Z^+ \\
m \in Z^+ \\
\begin{cases}
LL'_{ijn}(x) = LL_{ijn}(x) + L_{ij}, LL'_{ijn}(y) = LL_{ijn}(y) + W_{ij}, LL'_{ijn}(z) = LL_{ijn}(z) + H_{ij}, E_{ijn} = 1 \\
LL'_{ijn}(x) = LL_{ijn}(x) + W_{ij}, LL'_{ijn}(y) = LL_{ijn}(y) + L_{ij}, LL'_{ijn}(z) = LL_{ijn}(z) + H_{ij}, E_{ijn} = 2 \\
\vdots \\
LL'_{ijn}(x) = LL_{ijn}(x) + H_{ij}, LL'_{ijn}(y) = LL_{ijn}(y) + W_{ij}, LL'_{ijn}(z) = LL_{ijn}(z) + L_{ij}, E_{ijn} = 5 \\
LL'_{ijn}(x) = LL_{ijn}(x) + H_{ij}, LL'_{ijn}(y) = LL_{ijn}(y) + L_{ij}, LL'_{ijn}(z) = LL_{ijn}(z) + W_{ij}, E_{ijn} = 6
\end{cases} \\
LL'_{ijn}(x) \leqslant S_{1m}, LL'_{ijn}(y) \leqslant S_{2m}, LL'_{ijn}(z) \leqslant S_{3m},\ m \in [5,9] \\
\begin{cases}
S_{1m} + S_{3m} \geqslant \max\limits_{(i,j,n) \in U_{ikq}} \{LL'_{ijn}(x)\} + \max\limits_{(i,i,n) \in U_{ikq}} \{LL'_{ijn}(z)\} \\
S_{2m} + S_{3m} \geqslant \max\limits_{(i,j,n) \in U_{ikq}} \{LL'_{ijn}(y)\} + \max\limits_{(i,i,n) \in U_{ikq}} \{LL'_{ijn}(z)\}
\end{cases},\ m \in [1,4] \\
V_m = S_{1m} \cdot S_{2m} \cdot S_{3m} \\
\sum_{m=1}^{9} C_{im} \leqslant \sum_{j=1}^{J_i} N_{ij} \\
G(LL_{ijn}(x), LL_{ijn}(y)) \cdot G(LL_{ijn}(y), LL_{ijn}(z)) \cdot G(LL_{ijn}(z), LL_{ijn}(x)) = 1 \\
G(LL_{ijn}(x), LL_{ijn}(y)) = \begin{cases}
1, & \begin{cases}
\max(LL_{ij'n'}(x), LL_{ij''n''}(x)) \geqslant \min(LL'_{ij'n'}(x), LL'_{ij''n''}(x)) \\
and \\
\max(LL_{ij'n'}(y), LL_{ij''n''}(y)) \geqslant \min(LL'_{ij'n'}(y), LL'_{ij''n''}(y))
\end{cases} \\
0, else
\end{cases} \\
\min \sum_{i=1}^{4841} \sum_{m=1}^{9} C_{im} \\
LL_{ijn}(x) \geqslant 0, LL_{ijn}(y) \geqslant 0, LL_{ijn}(z) \geqslant 0
\end{cases} \tag{20}$$

6.1.3 物品装载模型的求解

●箱装模型的求解

箱装模型的具体求解步骤如下：

Step1： 输入箱子和物品的长、宽、高数据以及物品的数量和所属的订单。

Step2： 初始化物品和箱子列表。

Step3： 输入一个物品，判断该物品是否与之前的物品同属于一个订单，若是，则将物品相关数据增加至列表，重复执行 step3，否则执行 step4。

Step4： 根据从小到大的容积选择箱子对列表中的物品进行装载，同时更新列表为未装入箱子的物品信息。若不同容积的箱子装载同样大小和数量的物品，则选择容积最小的箱子。

Step5： 箱子的使用数量加 1。

Step6： 判断更新后的物品列表是否为空，若为空，则表示该订单中的物品都被装载完成，输出装载该订单所使用的箱子数量以及体积。

Step7： 重复 step2~step6，输出各订单所使用的箱子数量以及体积，计算箱子使用总数和总体积并输出。

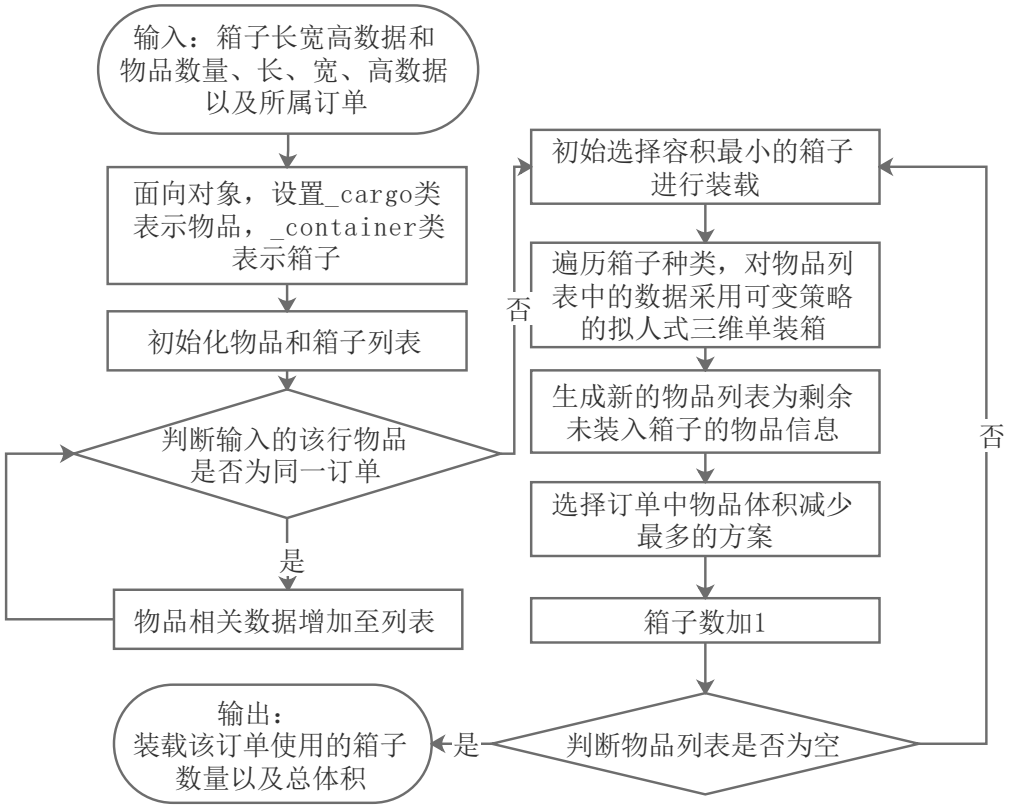

图 5：箱装模型求解算法流程图

得到箱装模型的求解结果如下：

表 1：箱装模型求解结果

箱子耗材使用总数	耗材总体积
8173	74064326000

下面，本文给出箱装具体方案的部分求解结果，完整结果见附件。

表 2：箱装方案部分求解结果

订单号	箱子数	箱子编号	耗材体积
1	1	5	10200000

2	4	5	40800000
⋮	⋮	⋮	⋮
5130	1	5	10200000
5131	1	3	4500000

以订单一为例，用 1 个 5 号自营纸箱即可完成订单 1 中物品的装载，订单 1 中物品的具体装载方案见下表：

表 3：订单 1 中物品具体装载方案

物品	x	y	z	物品长	物品宽	物品高
1	0	0	0	210	200	30
2	0	0	30	170	27	110
3	0	27	30	170	27	110
4	0	54	30	170	27	110
5	0	81	30	170	27	110
6	0	108	30	170	27	110
7	0	135	30	170	27	110
8	0	162	30	170	27	110

其中(x,y,z)表示物体最靠近原点的端点坐标，决定了物体的摆放位置。

可视化表达见下图：

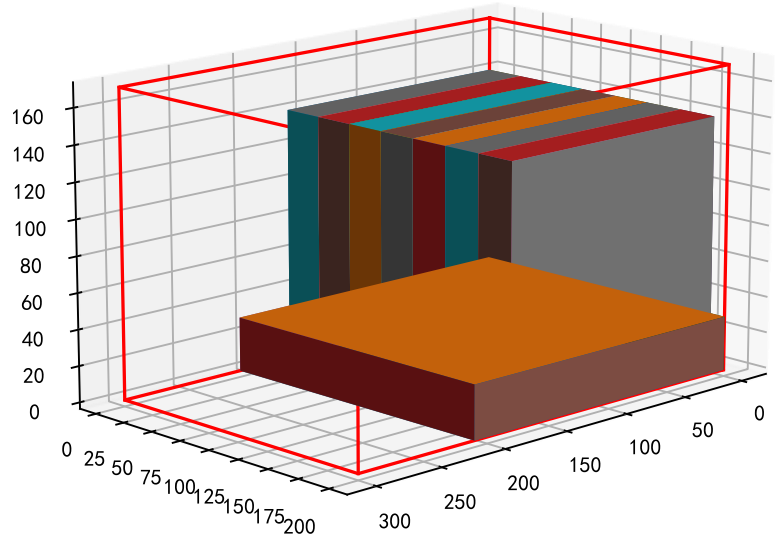

图 6：订单 1 物品箱装可视化图

●袋装模型的求解

袋装模型的求解步骤与箱装模型的求解步骤基本相同，主要改变了模型中的约束，求解得到的袋子耗材使用总数和袋子耗材使用总体积如下：

表 4：袋装模型求解结果

袋子耗材使用总数	耗材总体积
16442	2610000000

下面，本文给出具体袋装方案的部分求解结果，完整结果见附件。

表 5：袋装方案部分求解结果

订单号	袋子数	袋子编号	订单耗材体积
1	2	4	378000
2	3	4	567000
⋮	⋮	⋮	⋮

5131	1	2	75000
5132	2	2	150000

以订单一为例，用 1 个普通 4 号袋即可完成订单 1 中物品的装载，订单 1 中物品的具体装载方案见下表：

表 6：订单 1 中物品具体装载方案

物品	x	y	z	物品长	物品宽	物品高
1	0	0	0	30	200	210
2	30	0	0	27	110	170
3	57	0	0	27	110	170
4	84	0	0	27	110	170
5	111	0	0	27	110	170
6	138	0	0	27	110	170
7	165	0	0	27	110	170
8	192	0	0	27	110	170

其中(x,y,z)表示物体最靠近原点的端点坐标，决定了物体的摆放位置。

可视化表达见下图：

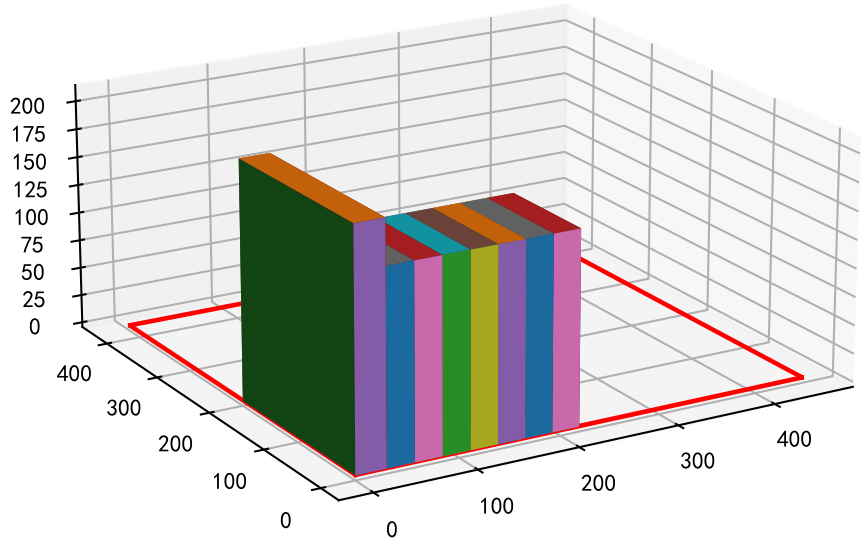

图 7：订单 1 物品袋装可视化图

●自由用材装载模型的求解

自由用材装载模型的求解步骤与上面基本相同，只需要分别考虑袋子和箱子装载物品的约束即可。

表 7：自由用材模型求解结果

耗材使用总数	耗材总体积
6309	13131824000

表 8：自由用材方案部分求解结果

订单号	箱子/袋子数	耗材编号	订单耗材体积
1	1	4	189000
2	3	4	567000
⋮	⋮	⋮	⋮
5131	1	2	75000
23	2	9	20400000

上表中，耗材编号 1~4 表示使用耗材种类为袋子，耗材编号 5~9 表示使用耗材种类

为箱子。

6.2 问题二模型的建立与求解

6.2.1 问题二的分析

问题二仅对耗材的尺寸进行优化，因此对问题一中建立的物品装载模型并没有太大的影响，仅需在模型中对耗材的长、宽、高赋予一个增量即可。

6.2.2 耗材尺寸优化模型

●目标函数

第一目标：优化后的方案所需耗材数量尽量少。假设优化后i订单使用第$m(m=1,2,\cdots,9)$种耗材的数量为C'_{im}。则目标的数学表达式为：

$$\min \sum_{i=1}^{5133}\sum_{m=1}^{9} C'_{im} \tag{21}$$

第二目标：优化后的方案所需耗材总体积越小越好。假设优化后第m种耗材的体积为V'_m，则目标的数学表达式为：

$$\min \sum_{i=1}^{5133}\sum_{m=1}^{9} C_{im}\cdot V'_m \tag{22}$$

●决策变量

决策变量为代表i订单j种类第n个物品不同长、宽、高组合的变量E_{ijn}，$1\leqslant E_{ijn}\leqslant 6$，$E_{ijn}\in Z^+$。

●约束条件

本题需要对耗材尺寸进行优化，假设S'_{1m}、S'_{2m}、S'_{3m}为优化后第m种耗材的长、宽、高，假设原耗材长、宽、高分别为S_{1m}、S_{2m}、S_{3m}，在原耗材长、宽、高上增设的增量为ΔS_{1m}、ΔS_{2m}、ΔS_{3m}，则有如下关系式：

$$\begin{cases} S'_{1m}=S_{1m}+\Delta S_{1m} \\ S'_{2m}=S_{2m}+\Delta S_{2m} \\ S'_{3m}=S_{3m}+\Delta S_{3m} \\ \Delta S_{1m}\in Z,\Delta S_{1m}\in Z,\Delta S_{1m}\in Z \\ S'_{1m},S'_{2m},S'_{3m}\geqslant 0 \end{cases} \tag{23}$$

题目要求优化方案所用耗材的总体积不能超过原方案的总体积，则有：

$$\sum_{i=1}^{5133}\sum_{m=1}^{9} C'_{im}S'_{1m}S'_{2m}S'_{3m} \leqslant \sum_{i=1}^{5133}\sum_{m=1}^{9} C_{im}S_{1m}S_{2m}S_{3m} \tag{24}$$

而装载约束与第一问中的物品装载模型相同。若$m\in[1,4]$, $m\in Z^+$，则约束条件与袋装模型的约束条件相同，为式(5)、(10)、(11)、(15)、(16)，若$m\in[5,9]$, $m\in Z^+$，则约束条件与箱装模型的约束条件相同，为式(5)、(6)、(7)、(10)、(11)，结合式(24)即可确定耗材优化模型的约束条件。

6.2.3 耗材尺寸优化模型的求解

本文在求解耗材尺寸优化模型过程中先对订单中的物体总体积进行聚类，将其聚成5类，再在这五类中，选取每类的订单最大体积，该类耗材的体积应大于等于该类订单的最大体积。通过循环搜索得到能够满足容积以及模型约束条件的耗材。

首先，以仅用箱子装载订单为例，需要把清洗后的数据进行订单体积的计算，然后

通过$k-means$方法将订单体积聚成 5 类，从而保证耗材尺寸优化后耗材的种类不变。聚类结果如图所示：

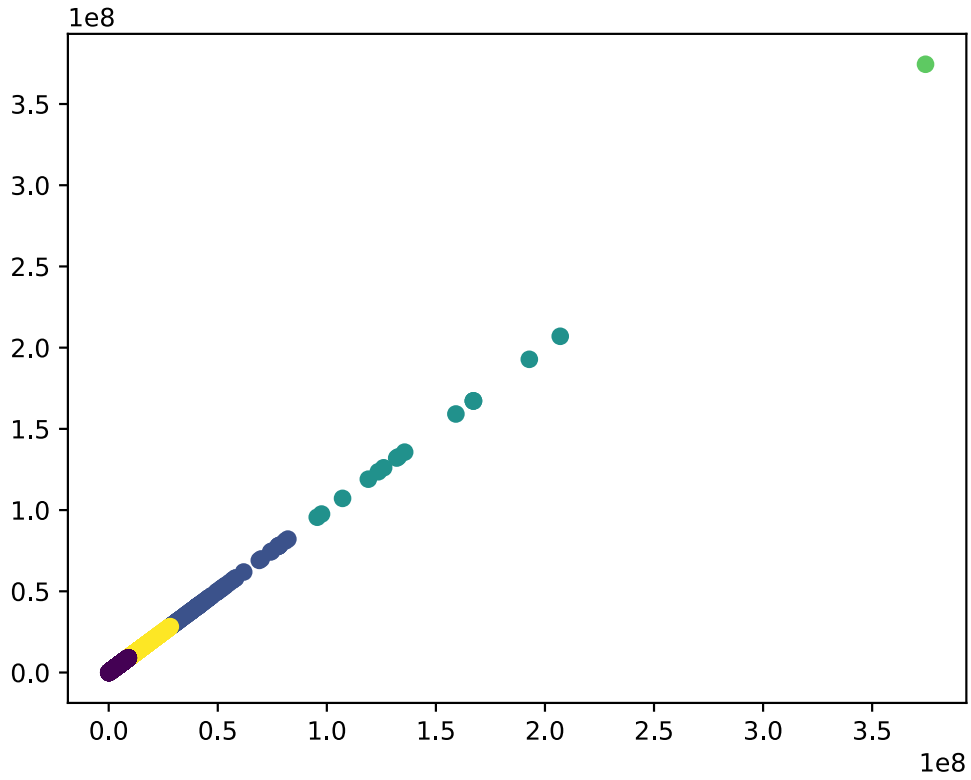

图 8：箱装订单体积聚类结果

聚类的部分结果见下表：

表 9：订单体积聚类部分结果

类别	订单编号
1	1，3，4，5，7，…
2	2，25，26，27，…
3	29，53，67，168，…
4	320，381，424，…
5	4040

本文采用逐步遍历法缩减搜索空间，逐步搜索更优解，求解耗材尺寸算法的主要步骤如下：

Step1：每一类中，选择最大的订单体积作为该类耗材应满足的容积。

Step2：对耗材的长、宽、高以 50 为步长搜索能够满足容积以及模型约束条件的耗材，在这些耗材中选择体积最小的耗材尺寸。

Step3：以 step2 中选出的耗材尺寸作为长、宽、高的中心进行进一步遍历，给定步长为 10，搜索能够满足容积以及模型约束条件的耗材，在这些耗材中选择体积最小的耗材尺寸。

Step4：以 step3 中选出的耗材尺寸作为长、宽、高的中心进行进一步遍历，给定步长为 1，搜索得到最终体积最小的耗材尺寸。

仅用箱装时求解得到的耗材尺寸结果见下表：

表 10：优化后箱子耗材尺寸

类别	长	宽	高
1	417	196	300
2	647	215	281
3	702	250	468
4	750	400	690
5	909	569	724

使用箱子耗材的总数为 4198，总体积为 126988550054。

仅用袋子装载情况下的聚类结果如下图所示：

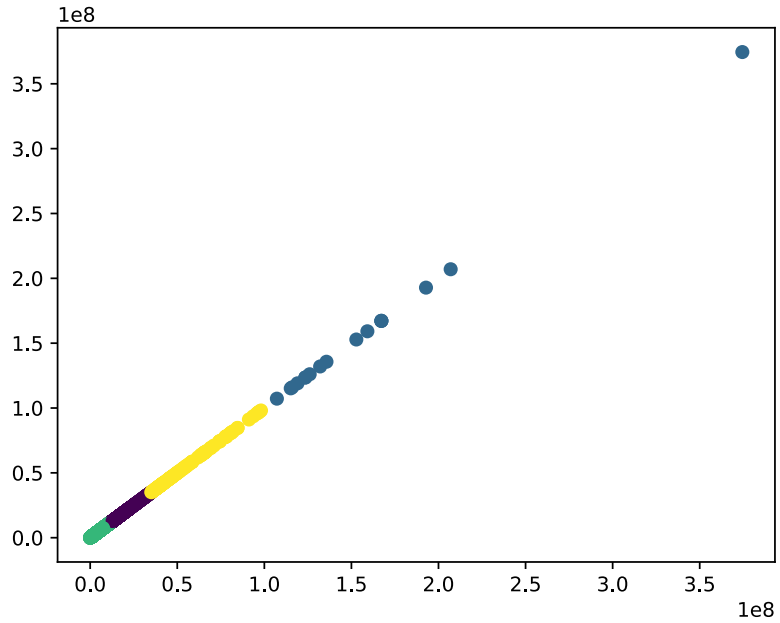

图 9：袋装订单体积聚类结果

采用逐步遍历法缩减搜索空间，逐步搜索更优解，求解得到耗材尺寸结果见下表：

表 11：优化后袋子耗材尺寸

类别	长	宽	高
1	713	492	4
2	919	487	9
3	1424	1074	16
4	1614	1274	19

6.3 问题三模型的建立与求解

6.3.1 问题三的分析

本题加入了耗材的柔性条件，在物品允许被轻微挤压的情况下，耗材能够进行伸展，并且伸展时的长、宽、高尺寸的变化不能超过原尺寸的 5%。在前两问的模型中将耗材的长、宽、高从定值改为变量，并对其进行 5%条件的约束添入约束条件内即可。在求解时，增加了对长、宽、高变量的循环遍历。

6.3.2 问题三模型的建立

在本题中，假设耗材的长、宽、高分别为 $\overline{S_{1m}}$ 、$\overline{S_{2m}}$ 、$\overline{S_{3m}}$，则根据耗材伸展不超过 5%的约束条件有：

$$\begin{cases} S_{1m} \leqslant \overline{S_{1m}} \leqslant 1.05 S_{1m} \\ S_{2m} \leqslant \overline{S_{2m}} \leqslant 1.05 S_{2m} \\ S_{3m} \leqslant \overline{S_{3m}} \leqslant 1.05 S_{3m} \end{cases} \tag{25}$$

在重新求解问题一和问题二的过程中，仅需将前面所建立的模型中的S_{1m}、S_{2m}、S_{3m}改为变量$\overline{S_{1m}}$、$\overline{S_{2m}}$、$\overline{S_{3m}}$即可，约束条件增加式(25)。

因此，对于问题一的箱装模型而言，目标函数为式(1)、(2)，决策变量为E_{ijn}、$\overline{S_{1m}}$、$\overline{S_{2m}}$、$\overline{S_{3m}}$，约束条件为式(5)、(6)、(7)、(10)、(11)、(25)；对问题一的袋装模型而言，目标函数为式(13)、(14)，决策变量不变，约束条件为式(5)、(10)、(11)、(15)、(16)、(25)；对问题一的自由用材装载模型而言，目标函数为式(18)、(19)，决策变量不变，约束条件为：若$m \in [1,4]$, $m \in Z^+$，则约束条件为式(5)、(10)、(11)、(15)、(16)、(25)，若$m \in [5,9]$, $m \in Z^+$，则约束条件为式(5)、(6)、(7)、(10)、(11)、(25)。

6.3.3 问题三模型的求解

本文在求解模型时先将耗材伸展至最大进行装载货物，尽可能保证耗材的使用数量最少，然后，根据物品距离原点最远端点的坐标对耗材的尺寸进行确定。算法的主要流程如下：

Step1： 令耗材的长、宽、高都伸展 5%，对物品进行装载。

Step2： 标记物品距离原点最远端点的坐标，将该坐标与耗材的长、宽、高比较，若比耗材的尺寸大，则选取对应坐标为耗材的长或宽或高，否则选择原本耗材的尺寸。

Step3： 输出耗材使用数量、尺寸和耗材总体积。

使用原耗材的情况下，求解结果为：

表 12：模型求解结果

箱装耗材使用总数	**箱装耗材总体积**
7608	388069000000
袋装耗材使用总数	**袋装耗材总体积**
16278	3210000000
自由用材耗材总数	**耗材总体积**
10523	301000000000

七、 模型的评价、改进与推广

模型优点

1.本文通过建立坐标系将空间中物品的位置通过点的坐标表示出来，巧妙地将耗材能否装载物品的问题转化为坐标与耗材长、宽、高的比较上，极大地简化了问题的复杂程度。

2.本文通过投影的方法将三维物品投影成二维平面判断空间中两物品是否存在重合，要求最终得到的结果严格满足这点要求，考虑较为周全，模型较为合理。

3.问题二的算法先对订单物品的总体积进行聚类，再确定耗材的尺寸，极大地降低了算法的时间复杂度，使得算法计算时间短，计算速度快。

模型缺点

本文建立的模型仅考虑了耗材的尺寸变化，并没有考虑物品被压缩时尺寸产生的变

化，如果考虑物品的柔性属性，将能够更加贴近实际情况。

模型推广

1.本文建立的模型具有较广的普适性，适用于各类三维装箱问题，能够在仓储、物流运输以及各类装箱领域根据不同的实际情况进行约束条件的调整。

2.本文建立的模型能够很好地帮助仓储进行耗材尺寸的选择，能够根据不同订单和物品确定不同的耗材尺寸，极大地减少了耗材成本和仓储成本。

八、 参考文献

[1]周光辉,仲邵伟,李邓宇卉,张毅祥.需求不完全拆分的多厢车辆路径和三维装箱模型与算法[J].管理评论,2022,34(08):299-312.DOI:10.14120/j.cnki.cn11-5057/f.2022.08.025.

[2] 智阳光. 三维装箱与选箱算法研究[D]. 西安电子科技大学,2021.DOI:10.27389/d.cnki.gxadu.2021.001844.

[3]丁纺,侯兆烽,赵凯芳.多种货物三维装箱问题研究[J].中国设备工程,2021(04):222-224.

[4]尚正阳,顾寄南,唐仕喜,孙晓红.高效求解三维装箱问题的剩余空间最优化算法[J].计算机工程与应用,2019,55(05):44-50.

[5]那日萨,崔雪莲,韩琪玮.基于实际约束的三维装箱问题优化算法[J].工业工程与管理,2017,22(04):10-16.DOI:10.19495/j.cnki.1007-5429.2017.04.002.

[6]雍文君. 多约束三维装箱问题的研究及实现[D].西安理工大学,2008.

[7]安吉振,陈衡,乔世超,潘佩媛,徐钢.基于 K-means 聚类算法的电站煤场来煤堆放优化研究[J].热力发电,2023,52(04):135-143.DOI:10.19666/j.rlfd.202206115.

[8]邱均平,孟炎镕.“双一流”高校科研成果质量研究——基于 K-Means 聚类和 Logistic 回归分析[J].图书馆理论与实践,2021(05):9-15.DOI:10.14064/j.cnki.issn1005-8214.2021.05.001.

九、 附录

代码：第一问

问题一_箱子：

```
from typing import Iterable, List
from Encase3D._cargo import *
from Encase3D._container import *

class Strategy(object):
    # 继承此类 重写两个静态函数 实现自定义两个装载策略: 装箱顺序 和 货物.
    @staticmethod
    def encasement_sequence(cargos:Iterable) -> Iterable:
        return cargos

    @staticmethod
    def choose_cargo_poses(cargo:Cargo, container:Container) -> list:
        return list(CargoPose)

def encase_cargos_into_container(
    cargos:Iterable,
    container:Container,
    strategy:type
) -> float:
    sorted_cargos:List[Cargo] = strategy.encasement_sequence(cargos)
    i = 0 # 记录发当前货物
    while i < len(sorted_cargos):
        j = 0 # 记录当前摆放方式
        cargo = sorted_cargos[i]
        poses = strategy.choose_cargo_poses(cargo, container)
        while j < len(poses):
            cargo.pose = poses[j]
            is_encased = container._encase(cargo)
            if is_encased.is_valid:
                break # 可以装入 不在考虑后续摆放方式
            j += 1   # 不可装入 查看下一个摆放方式
        if is_encased.is_valid:
            i += 1 # 成功放入 继续装箱
        elif is_encased == Point(-1,-1,0):
            continue # 没放进去但是修改了参考面位置 重装
        else :
            i += 1 # 纯纯没放进去 跳过看下一个箱子
```

```
    return sum(list(map(
                lambda cargo:cargo.volume,container._setted_cargos
            ))) / container.volume
def failed_encase_cargos_into_container(
    cargos: Iterable, # 货物列表，包含所有要装箱的货物
    container: Container, # 集装箱对象
    strategy: type # 装箱策略，必须是 BaseEncasementStrategy 的子类
) -> List[Cargo]:
    """
    将一批货物尽可能地装入集装箱中，并返回未能装入集装箱的货物列表。

    Args:
        cargos (Iterable): 货物列表，包含所有要装箱的货物
        container (Container): 集装箱对象，用于盛放货物
        strategy (type): 装箱策略，必须是 BaseEncasementStrategy 的子类

    Returns:
        List[Cargo]: 未能装入集装箱的货物列表
    """

    # 通过调用装箱策略的方法获取排序后的货物列表
    sorted_cargos:List[Cargo] = strategy.encasement_sequence(cargos)

    i = 0 # 记录当前装箱的货物序号
    failed_cargos = [] # 记录未能装入集装箱的货物列表
    while i < len(sorted_cargos):
        j = 0 # 记录当前摆放方式的序号
        cargo = sorted_cargos[i]

        # 获取当前货物在集装箱中所有可摆放的位置
        poses = strategy.choose_cargo_poses(cargo, container)

        while j < len(poses):
            cargo.pose = poses[j] # 将货物摆放在当前位置
            is_encased = container._encase(cargo) # 尝试将货物装入集装箱

            if is_encased.is_valid:
                break # 可以装入，不再考虑后续摆放方式

            j += 1 # 不可装入，查看下一个摆放方式

        if is_encased.is_valid:
```

```
                i += 1 # 当前货物已成功放置，继续装箱
            elif is_encased == Point(-1,-1,0):
                continue # 没有放进去，但是修改了参考面位置，重装
            else :
                failed_cargos.append(cargo) # 将未能装入集装箱的货物添加到列表中
                i += 1 # 跳过看下一个货物

    # 返回未能装入集装箱的货物列表
    return failed_cargos
def yanzhanfailed_encase_cargos_into_container(
    cargos: Iterable, # 货物列表，包含所有要装箱的货物
    container: Container, # 集装箱对象
    strategy: type # 装箱策略，必须是 BaseEncasementStrategy 的子类
) -> List[Cargo]:
    """
    将一批货物尽可能地装入集装箱中，并返回未能装入集装箱的货物列表。

    Args:
        cargos (Iterable): 货物列表，包含所有要装箱的货物
        container (Container): 集装箱对象，用于盛放货物
        strategy (type): 装箱策略，必须是 BaseEncasementStrategy 的子类

    Returns:
        List[Cargo]: 未能装入集装箱的货物列表
    """

    # 通过调用装箱策略的方法获取排序后的货物列表
    sorted_cargos:List[Cargo] = strategy.encasement_sequence(cargos)

    i = 0 # 记录当前装箱的货物序号
    failed_cargos = [] # 记录未能装入集装箱的货物列表
    while i < len(sorted_cargos):
        j = 0 # 记录当前摆放方式的序号
        cargo = sorted_cargos[i]

        # 获取当前货物在集装箱中所有可摆放的位置
        poses = strategy.choose_cargo_poses(cargo, container)

        while j < len(poses):
            cargo.pose = poses[j] # 将货物摆放在当前位置
            is_encased = container.yanzhan_encase(cargo) # 尝试将货物装入集装箱
```

```
                if is_encased.is_valid:
                    break # 可以装入，不再考虑后续摆放方式

                j += 1 # 不可装入，查看下一个摆放方式

            if is_encased.is_valid:
                i += 1 # 当前货物已成功放置，继续装箱
            elif is_encased == Point(-1,-1,0):
                continue # 没有放进去，但是修改了参考面位置，重装
            else :
                failed_cargos.append(cargo) # 将未能装入集装箱的货物添加到列表中
                i += 1 # 跳过看下一个货物

    # 返回未能装入集装箱的货物列表
    return failed_cargos
#daizi_encase
def daizifailed_encase_cargos_into_container(
    cargos: Iterable, # 货物列表，包含所有要装箱的货物
    container: Container, # 集装箱对象
    strategy: type # 装箱策略，必须是 BaseEncasementStrategy 的子类
) -> List[Cargo]:
    """
    将一批货物尽可能地装入集装箱中，并返回未能装入集装箱的货物列表。

    Args:
        cargos (Iterable): 货物列表，包含所有要装箱的货物
        container (Container): 集装箱对象，用于盛放货物
        strategy (type): 装箱策略，必须是 BaseEncasementStrategy 的子类

    Returns:
        List[Cargo]: 未能装入集装箱的货物列表
    """

    # 通过调用装箱策略的方法获取排序后的货物列表
    sorted_cargos:List[Cargo] = strategy.encasement_sequence(cargos)

    i = 0 # 记录当前装箱的货物序号
    failed_cargos = [] # 记录未能装入集装箱的货物列表
    while i < len(sorted_cargos):
        j = 0 # 记录当前摆放方式的序号
        cargo = sorted_cargos[i]
```

```
            # 获取当前货物在集装箱中所有可摆放的位置
            poses = strategy.choose_cargo_poses(cargo, container)

            while j < len(poses):
                cargo.pose = poses[j] # 将货物摆放在当前位置
                is_encased = container.daizi_encase(cargo) # 尝试将货物装入集装箱

                if is_encased.is_valid:
                    break # 可以装入，不再考虑后续摆放方式

                j += 1 # 不可装入，查看下一个摆放方式

            if is_encased.is_valid:
                i += 1 # 当前货物已成功放置，继续装箱
            elif is_encased == Point(-1,-1,0):
                continue # 没有放进去，但是修改了参考面位置，重装
            else :
                failed_cargos.append(cargo) # 将未能装入集装箱的货物添加到列表中
                i += 1 # 跳过看下一个货物

    # 返回未能装入集装箱的货物列表
    return failed_cargos
def daizifailed_encase_cargos_into_container(
    cargos: Iterable, # 货物列表，包含所有要装箱的货物
    container: Container, # 集装箱对象
    strategy: type # 装箱策略，必须是 BaseEncasementStrategy 的子类
) -> List[Cargo]:
    """
    将一批货物尽可能地装入集装箱中，并返回未能装入集装箱的货物列表。

    Args:
        cargos (Iterable): 货物列表，包含所有要装箱的货物
        container (Container): 集装箱对象，用于盛放货物
        strategy (type): 装箱策略，必须是 BaseEncasementStrategy 的子类

    Returns:
        List[Cargo]: 未能装入集装箱的货物列表
    """

    # 通过调用装箱策略的方法获取排序后的货物列表
    sorted_cargos:List[Cargo] = strategy.encasement_sequence(cargos)
```

```
    i = 0 # 记录当前装箱的货物序号
    failed_cargos = [] # 记录未能装入集装箱的货物列表
    while i < len(sorted_cargos):
        j = 0 # 记录当前摆放方式的序号
        cargo = sorted_cargos[i]

        # 获取当前货物在集装箱中所有可摆放的位置
        poses = strategy.choose_cargo_poses(cargo, container)

        while j < len(poses):
            cargo.pose = poses[j] # 将货物摆放在当前位置
            is_encased = container.daizi_encase(cargo) # 尝试将货物装入集装箱

            if is_encased.is_valid:
                break # 可以装入，不再考虑后续摆放方式

            j += 1 # 不可装入，查看下一个摆放方式

        if is_encased.is_valid:
            i += 1 # 当前货物已成功放置，继续装箱
        elif is_encased == Point(-1,-1,0):
            continue # 没有放进去，但是修改了参考面位置，重装
        else :
            failed_cargos.append(cargo) # 将未能装入集装箱的货物添加到列表中
            i += 1 # 跳过看下一个货物

    # 返回未能装入集装箱的货物列表
    return failed_cargos
def yanzhandaizifailed_encase_cargos_into_container(
    cargos: Iterable, # 货物列表，包含所有要装箱的货物
    container: Container, # 集装箱对象
    strategy: type # 装箱策略，必须是 BaseEncasementStrategy 的子类
) -> List[Cargo]:
    """
    将一批货物尽可能地装入集装箱中，并返回未能装入集装箱的货物列表。

    Args:
        cargos (Iterable): 货物列表，包含所有要装箱的货物
        container (Container): 集装箱对象，用于盛放货物
        strategy (type): 装箱策略，必须是 BaseEncasementStrategy 的子类

    Returns:
```

```
            List[Cargo]:  未能装入集装箱的货物列表
        """

        # 通过调用装箱策略的方法获取排序后的货物列表
        sorted_cargos:List[Cargo] = strategy.encasement_sequence(cargos)

        i = 0 #  记录当前装箱的货物序号
        failed_cargos = [] #  记录未能装入集装箱的货物列表
        while i < len(sorted_cargos):
            j = 0 #  记录当前摆放方式的序号
            cargo = sorted_cargos[i]

            # 获取当前货物在集装箱中所有可摆放的位置
            poses = strategy.choose_cargo_poses(cargo, container)

            while j < len(poses):
                cargo.pose = poses[j] # 将货物摆放在当前位置
                is_encased = container.yanzhandaizi_encase(cargo) #  尝试将货物装入集装箱

                if is_encased.is_valid:
                    break #  可以装入，不再考虑后续摆放方式

                j += 1 #  不可装入，查看下一个摆放方式

            if is_encased.is_valid:
                i += 1 #  当前货物已成功放置，继续装箱
            elif is_encased == Point(-1,-1,0):
                continue #  没有放进去，但是修改了参考面位置，重装
            else :
                failed_cargos.append(cargo) # 将未能装入集装箱的货物添加到列表中
                i += 1 #  跳过看下一个货物

        # 返回未能装入集装箱的货物列表
        return failed_cargos
class VolumeGreedyStrategy(Strategy):
    @staticmethod
    def encasement_sequence(cargos:Iterable) -> Iterable:
        return sorted(cargos, key= lambda cargo:cargo.volume,reverse=1)

    @staticmethod
    def choose_cargo_poses(cargo:Cargo, container:Container) -> list:
        return list(CargoPose)
```

```
from enum import Enum


class CargoPose(Enum):  # 定义枚举类型，用于描述货物的不同摆放方式
    tall_wide = 0
    tall_thin = 1
    mid_wide = 2
    mid_thin = 3
    short_wide = 4
    short_thin = 5


class Point(object):  # 定义一个坐标点类，表示三维空间中的一个点
    def __init__(self, x: int, y: int, z: int) -> None:  # Point 类的构造函数，初始化 x、y、z 三个坐标值
        self.x = x
        self.y = y
        self.z = z

    def __repr__(self) -> str:  # 将 Point 对象转换成字符串表示形式
        return f"({self.x},{self.y},{self.z})"

    def __eq__(self, __o: object) -> bool:  # 重载==操作符，用于比较两个 Point 对象是否相等
        return self.x == __o.x and self.y == __o.y and self.z == __o.z

    @property
    def is_valid(self) -> bool:  # 判断坐标点是否合法（即坐标值都是非负整数）
        return self.x >= 0 and self.y >= 0 and self.z >= 0

    @property
    def tuple(self) -> tuple:  # 将 Point 对象转换为元组
        return (self.x, self.y, self.z)


class Cargo(object):  # 定义货物类
    def __init__(self, length: int, width: int, height: int) -> None:  # Cargo 类的构造函数，初始化货物的长、宽、高
        self._point = Point(-1, -1, -1)  # 表示货物的坐标点（初始值为-1，表示未确定）
        # 表示货物的形状（用集合来存储，即使属性_shape 被定义为集合类型，也应该注意到集合类型是无序的。因此，当你创建 Cargo 对象时，属性_shape 中元素的顺序是不确定的）
```

```
        self._shape = {length, width,height}
        self._pose = CargoPose.tall_thin   # 表示货物的摆放方式（默认为 tall_thin）

    def is_setted(self):
        return self.x >= 0

    def __repr__(self) -> str:   # 将 Cargo 对象转换成字符串表示形式
        return f"{self._point} {self.shape}"

    @property
    def pose(self) -> CargoPose:   # 获取货物的摆放方式
        return self._pose

    @pose.setter
    def pose(self, new_pose: CargoPose):   # 设置货物的摆放方式
        self._pose = new_pose

    @property
    def _shape_swiche(self) -> dict:   # 根据不同的摆放方式，返回货物的形状（长、宽、高）
        edges = sorted(self._shape)
        return {
            CargoPose.tall_thin: (edges[1], edges[0], edges[-1]),
            CargoPose.tall_wide: (edges[0], edges[1], edges[-1]),
            CargoPose.mid_thin: (edges[-1], edges[0], edges[1]),
            CargoPose.mid_wide: (edges[0], edges[-1], edges[-1]),
            CargoPose.short_thin: (edges[-1], edges[1], edges[0]),
            CargoPose.short_wide: (edges[1], edges[-1], edges[0])
        }

    @property
    def point(self):   # 获取货物的坐标点
        return self._point

    @point.setter
    def point(self, new_point: Point):   # 设置货物的坐标点
        self._point = new_point

    @property
    def x(self) -> int:   # 获取货物的 x 坐标
        return self._point.x

    @x.setter
```

```
    def x(self, new_x: int):    # 设置货物的 x 坐标
        self._point = Point(new_x, self.y, self.z)

    @property
    def y(self) -> int:    # 获取货物的 y 坐标
        return self._point.y

    @y.setter
    def y(self, new_y: int):    # 设置货物的 y 坐标
        self._point = Point(self.x, new_y, self.z)

    @property
    def z(self) -> int:    # 获取货物的 z 坐标
        return self._point.z

    @z.setter
    def z(self, new_z: int):    # 设置货物的 z 坐标
        self._point = Point(self.z, self.y, new_z)

    @property
    def length(self) -> int:    # 获取货物的长
        return self.shape[0]

    @property
    def width(self) -> int:    # 获取货物的宽
        return self.shape[1]

    @property
    def height(self) -> int:    # 获取货物的高
        return self.shape[-1]

    def get_shadow_of(self, planar: str) -> tuple:    # 返回货物在某个平面上的投影区域
        if planar in ("xy", "yx"):
            x0, y0 = self.x, self.y
            x1, y1 = self.x + self.length, self.y + self.width
        elif planar in ("xz", "zx"):
            x0, y0 = self.x, self.z
            x1, y1 = self.x + self.length, self.z + self.height
        elif planar in ("yz", "zy"):
            x0, y0 = self.y, self.z
            x1, y1 = self.y + self.width, self.z + self.height
        return (x0, y0, x1, y1)
```

```
    @property
    def shape(self) -> tuple:   # 获取货物的形状（长、宽、高）
        return self._shape_swiche[self._pose]

    @shape.setter
    def shape(self, length, width, height):   # 设置货物的形状（长、宽、高）
        self._shape = {length, width, height}

    @property
    def volume(self) -> int:   # 计算货物的体积
        reslut = 1
        for i in self._shape:
            reslut *= i
        return reslut
from time import time
from typing import List
from Encase3D._cargo import *
from copy import deepcopy


class Container(object):
    def __init__(self, length: int, width: int, height: int) -> None:
        self._length = length
        self._width = width
        self._height = height
        self._refresh()
        self.maxx = 0
        self.maxy = 0
        self.maxz = 0
    def returnv(self):
        return max(self._length,self.maxx)*max(self._width,self.maxy)*max(self._height,self.maxz)
    def __repr__(self) -> str:
        return f"{self._length}, {self._width}, {self._height}"

    def _refresh(self):
        self._horizontal_planar = 0   # 水平放置参考面
        self._vertical_planar = 0   # 垂直放置参考面
        self._available_points = [Point(0, 0, 0)]   # 可放置点有序集合
        self._setted_cargos : List[Cargo] = []

    def _sort_available_points(self):
```

```
        self._available_points.sort(key=lambda x: x.z)
        self._available_points.sort(key=lambda x: x.x)
        self._available_points.sort(key=lambda x: x.y)

    def is_encasable(self, site: Point, cargo: Cargo) -> bool:
        encasable = True
        temp = deepcopy(cargo)
        temp.point = site
        if (
            temp.x + temp.length > self.length or
            temp.y + temp.width > self.width or
            temp.z + temp.height > self.height
        ):
            encasable = False
        for setted_cargo in self._setted_cargos:
            if _is_cargos_collide(temp, setted_cargo):
                encasable = False
        return encasable
    def yanzhanis_encasable(self, site: Point, cargo: Cargo) -> bool:
        encasable = True
        temp = deepcopy(cargo)
        temp.point = site
        if self.maxx <= temp.x + temp.length:
            self.maxx = temp.x + temp.length
        if self.maxy <= temp.y + temp.width:
            self.maxy = temp.y + temp.width
        if self.maxz <= temp.z + temp.height:
            self.maxz = temp.z + temp.height
        if (
            temp.x + temp.length > self.length*1.05 or
            temp.y + temp.width > self.width*1.05 or
            temp.z + temp.height > self.height*1.05
        ):
            encasable = False
        for setted_cargo in self._setted_cargos:
            if _is_cargos_collide(temp, setted_cargo):
                encasable = False
        return encasable
    def _encase(self, cargo: Cargo) -> Point:
        # flag 存储放置位置, (-1,-1,0)放置失败并调整参考面, (-1,-1,-1)放置失败.
        flag = Point(-1, -1, -1)
        # 用于记录执行前的参考面位置，便于后续比较
```

```
        history = [self._horizontal_planar, self._vertical_planar]

        def __is_planar_changed() -> bool:
            return (
                    not flag.is_valid and   # 防止破坏已经确定可放置的点位, 即只能在(-1, -1, -1)基础上改
                    self._horizontal_planar == history[0] and
                    self._vertical_planar == history[-1]
            )
            # 遍历所有可用点位

        for point in self._available_points:
            if (
                    self.is_encasable(point, cargo) and   # 判断当前点是否能放置货物
                    point.x + cargo.length < self._horizontal_planar and   # 判断横坐标是否超出边界
                    point.z + cargo.height < self._vertical_planar   # 判断纵坐标是否超出边界
            ):
                flag = point
                break   # 找到一个符合要求的点就将其标记为待放置点并中断循环
        # 如果找不到可用点位，则先检查当前参考面是否位于横轴边界上
        if not flag.is_valid:
            if (
                    self._horizontal_planar == 0 or
                    self._horizontal_planar == self.length
            ):
                # 在 yz 坐标为 (0, self._vertical_planar) 的点位上尝试放置货物
                if self.is_encasable(Point(0, 0, self._vertical_planar), cargo):
                    flag = Point(0, 0, self._vertical_planar)
                    self._vertical_planar += cargo.height   # 向上升高参考面
                    self._horizontal_planar = cargo.length   # 向右移动参考面
                elif self._vertical_planar < self.height:   # 如果不能在边界上放置，就向上升高参考面
                    self._vertical_planar = self.height
                    self._horizontal_planar = self.length
                    if __is_planar_changed():
                        flag.z == 0   # 放置失败并调整参考面
            else:
                # 在同一水平位置上寻找底部空位
                for point in self._available_points:
                    if (
                            point.x == self._horizontal_planar and
```

```
                                point.y == 0 and
                                self.is_encasable(point, cargo) and
                                point.z + cargo.height <= self._vertical_planar
                        ):
                            flag = point
                            self._horizontal_planar += cargo.length  # 向右移动参考面
                            break
                    if not flag.is_valid:
                        self._horizontal_planar = self.length
                        if __is_planar_changed():
                            flag.z == 0  # 放置失败并调整参考面
            # 如果找到可用点位，则进行相应的操作并更新货物的位置信息、可用点集合和已摆放货物数量
            if flag.is_valid:
                cargo.point = flag
                if flag in self._available_points:
                    self._available_points.remove(flag)
                self._adjust_setting_cargo(cargo)
                self._setted_cargos.append(cargo)
                self._available_points.extend([
                    Point(cargo.x + cargo.length, cargo.y, cargo.z),
                    Point(cargo.x, cargo.y + cargo.width, cargo.z),
                    Point(cargo.x, cargo.y, cargo.z + cargo.height)
                ])
                self._sort_available_points()
            return flag
        def yanzhan_encase(self, cargo: Cargo) -> Point:
            # flag 存储放置位置, (-1,-1,0)放置失败并调整参考面, (-1,-1,-1)放置失败.
            flag = Point(-1, -1, -1)
            # 用于记录执行前的参考面位置, 便于后续比较
            history = [self._horizontal_planar, self._vertical_planar]

            def __is_planar_changed() -> bool:
                return (
                        not flag.is_valid and  # 防止破坏已经确定可放置的点位, 即只能在(-1, -1, -1)基础上改
                        self._horizontal_planar == history[0] and
                        self._vertical_planar == history[-1]
                )
                # 遍历所有可用点位

            for point in self._available_points:
```

```
                if (
                        self.yanzhanis_encasable(point, cargo) and   # 判断当前点是否能放置货物
                        point.x + cargo.length < self._horizontal_planar*1.05 and   # 判断横坐标是否超出边界
                        point.z + cargo.height < self._vertical_planar*1.05   # 判断纵坐标是否超出边界
                ):
                    flag = point
                    break   # 找到一个符合要求的点就将其标记为待放置点并中断循环
            # 如果找不到可用点位，则先检查当前参考面是否位于横轴边界上
            if not flag.is_valid:
                if (
                        self._horizontal_planar == 0 or
                        self._horizontal_planar == self.length
                ):
                    # 在 yz 坐标为 (0, self._vertical_planar) 的点位上尝试放置货物
                    if self.yanzhanis_encasable(Point(0, 0, self._vertical_planar), cargo):
                        flag = Point(0, 0, self._vertical_planar)
                        self._vertical_planar += cargo.height   # 向上升高参考面
                        self._horizontal_planar = cargo.length   # 向右移动参考面
                    elif self._vertical_planar < self.height:   # 如果不能在边界上放置，就向上升高参考面
                        self._vertical_planar = self.height
                        self._horizontal_planar = self.length
                        if __is_planar_changed():
                            flag.z == 0   # 放置失败并调整参考面
                else:
                    # 在同一水平位置上寻找底部空位
                    for point in self._available_points:
                        if (
                                point.x == self._horizontal_planar and
                                point.y == 0 and
                                self.yanzhanis_encasable(point, cargo) and
                                point.z + cargo.height <= self._vertical_planar
                        ):
                            flag = point
                            self._horizontal_planar += cargo.length   # 向右移动参考面
                            break
                    if not flag.is_valid:
                        self._horizontal_planar = self.length
                        if __is_planar_changed():
                            flag.z == 0   # 放置失败并调整参考面
```

```
        # 如果找到可用点位，则进行相应的操作并更新货物的位置信息、可用点集合和已摆放货物数量
        if flag.is_valid:
            cargo.point = flag
            if flag in self._available_points:
                self._available_points.remove(flag)
            self._adjust_setting_cargo(cargo)
            self._setted_cargos.append(cargo)
            self._available_points.extend([
                Point(cargo.x + cargo.length, cargo.y, cargo.z),
                Point(cargo.x, cargo.y + cargo.width, cargo.z),
                Point(cargo.x, cargo.y, cargo.z + cargo.height)
            ])
            self._sort_available_points()
        return flag
    def daiziis_encasable(self, site: Point, cargo: Cargo) -> bool:
        encasable = True
        temp = deepcopy(cargo)
        temp.point = site


        if self.maxx <= temp.x + temp.length:
            self.maxx = temp.x + temp.length
        if self.maxy <= temp.y + temp.width:
            self.maxy = temp.y + temp.width
        if (
            self.maxx+temp.height+temp.z > self.length+1 or ## 袋子长+袋子高 >= 物品长+物品高
            self.maxy + temp.height+temp.z> self.width + 1 ## 袋子宽+袋子高 >= 物品宽+物品高
        ):
            encasable = False
        for setted_cargo in self._setted_cargos:
            if _is_cargos_collide(temp, setted_cargo):
                encasable = False
        return encasable
    def yanzhandaiziis_encasable(self, site: Point, cargo: Cargo) -> bool:
        encasable = True
        temp = deepcopy(cargo)
        temp.point = site


        if self.maxx <= temp.x + temp.length:
```

```
                self.maxx = temp.x + temp.length
            if self.maxy <= temp.y + temp.width:
                self.maxy = temp.y + temp.width
            if self.maxz <= temp.z + temp.height:
                self.maxz = temp.z + temp.height
            if (
                self.maxx+temp.height+temp.z > (self.length+1)*1.05 or ##  袋子长+袋子高  >=  物品长
+物品高
                self.maxy + temp.height+temp.z> (self.width + 1)*1.05 ##  袋子宽+袋子高  >=  物品宽+
物品高
            ):
                encasable = False
            for setted_cargo in self._setted_cargos:
                if _is_cargos_collide(temp, setted_cargo):
                    encasable = False
            return encasable

        def daizi_encase(self, cargo: Cargo) -> Point:
            # flag 存储放置位置, (-1,-1,0)放置失败并调整参考面, (-1,-1,-1)放置失败.
            flag = Point(-1, -1, -1)
            #  用于记录执行前的参考面位置, 便于后续比较
            history = [self._horizontal_planar, self._vertical_planar]

            def __is_planar_changed() -> bool:
                return (
                        not flag.is_valid and    #  防止破坏已经确定可放置的点位, 即只能在(-1, -1, -
1)基础上改
                        self._horizontal_planar == history[0] and
                        self._vertical_planar == history[-1]
                )
                #  遍历所有可用点位

            for point in self._available_points:
                if (
                        self.daiziis_encasable(point, cargo) and    #  判断当前点是否能放置货物
                        point.x + cargo.length + cargo.height< self._horizontal_planar +1 and    #  判断
横坐标是否超出边界
                        point.z + cargo.width + cargo.height < self._vertical_planar+1    #  判断纵坐标
是否超出边界
                ):
                    flag = point
                    break    #  找到一个符合要求的点就将其标记为待放置点并中断循环
```

```
        # 如果找不到可用点位，则先检查当前参考面是否位于横轴边界上
        if not flag.is_valid:
            if (
                    self._horizontal_planar == 0 or
                    self._horizontal_planar == self.length
            ):
                # 在 yz 坐标为 (0, self._vertical_planar) 的点位上尝试放置货物
                if self.daiziis_encasable(Point(0, 0, self._vertical_planar), cargo):
                    flag = Point(0, 0, self._vertical_planar)
                    self._vertical_planar += cargo.height  # 向上升高参考面
                    self._horizontal_planar = cargo.length  # 向右移动参考面
                elif self._vertical_planar < self.height:  # 如果不能在边界上放置，就向上升高参考面
                    self._vertical_planar = self.height
                    self._horizontal_planar = self.length
                    if __is_planar_changed():
                        flag.z == 0  # 放置失败并调整参考面
            else:
                # 在同一水平位置上寻找底部空位
                for point in self._available_points:
                    if (
                            point.x == self._horizontal_planar and
                            point.y == 0 and
                            self.daiziis_encasable(point, cargo) and
                            point.z + cargo.height <= self._vertical_planar
                    ):
                        flag = point
                        self._horizontal_planar += cargo.length  # 向右移动参考面
                        break
                if not flag.is_valid:
                    self._horizontal_planar = self.length
                    if __is_planar_changed():
                        flag.z == 0  # 放置失败并调整参考面
        # 如果找到可用点位，则进行相应的操作并更新货物的位置信息、可用点集合和已摆放货物数量
        if flag.is_valid:
            cargo.point = flag
            if flag in self._available_points:
                self._available_points.remove(flag)
            self._adjust_setting_cargo(cargo)
            self._setted_cargos.append(cargo)
            self._available_points.extend([
```

```
                    Point(cargo.x + cargo.length, cargo.y, cargo.z),
                    Point(cargo.x, cargo.y + cargo.width, cargo.z),
                    Point(cargo.x, cargo.y, cargo.z + cargo.height)
                ])
                self._sort_available_points()
        return flag
    def yanzhandaizi_encase(self, cargo: Cargo) -> Point:
        # flag 存储放置位置, (-1,-1,0)放置失败并调整参考面, (-1,-1,-1)放置失败.
        flag = Point(-1, -1, -1)
        # 用于记录执行前的参考面位置, 便于后续比较
        history = [self._horizontal_planar, self._vertical_planar]

        def __is_planar_changed() -> bool:
            return (
                    not flag.is_valid and   # 防止破坏已经确定可放置的点位, 即只能在(-1, -1, -1)基础上改
                    self._horizontal_planar == history[0] and
                    self._vertical_planar == history[-1]
            )
            # 遍历所有可用点位

        for point in self._available_points:
            if (
                    self.daiziis_encasable(point, cargo) and   # 判断当前点是否能放置货物
                    point.x + cargo.length + cargo.height< (self._horizontal_planar +1)*1.05 and   # 判断横坐标是否超出边界
                    point.z + cargo.width + cargo.height < (self._vertical_planar+1)*1.05    # 判断纵坐标是否超出边界
            ):
                flag = point
                break   # 找到一个符合要求的点就将其标记为待放置点并中断循环
        # 如果找不到可用点位，则先检查当前参考面是否位于横轴边界上
        if not flag.is_valid:
            if (
                    self._horizontal_planar == 0 or
                    self._horizontal_planar == self.length
            ):
                # 在 yz 坐标为 (0, self._vertical_planar) 的点位上尝试放置货物
                if self.daiziis_encasable(Point(0, 0, self._vertical_planar), cargo):
                    flag = Point(0, 0, self._vertical_planar)
                    self._vertical_planar += cargo.height   # 向上升高参考面
                    self._horizontal_planar = cargo.length   # 向右移动参考面
```

```
                elif self._vertical_planar < self.height:  # 如果不能在边界上放置，就向上升高参考面
                    self._vertical_planar = self.height
                    self._horizontal_planar = self.length
                    if __is_planar_changed():
                        flag.z == 0  # 放置失败并调整参考面
            else:
                # 在同一水平位置上寻找底部空位
                for point in self._available_points:
                    if (
                            point.x == self._horizontal_planar and
                            point.y == 0 and
                            self.daiziis_encasable(point, cargo) and
                            point.z + cargo.height <= self._vertical_planar
                    ):
                        flag = point
                        self._horizontal_planar += cargo.length  # 向右移动参考面
                        break
                if not flag.is_valid:
                    self._horizontal_planar = self.length
                    if __is_planar_changed():
                        flag.z == 0  # 放置失败并调整参考面
        # 如果找到可用点位，则进行相应的操作并更新货物的位置信息、可用点集合和已摆放货物数量
        if flag.is_valid:
            cargo.point = flag
            if flag in self._available_points:
                self._available_points.remove(flag)
            self._adjust_setting_cargo(cargo)
            self._setted_cargos.append(cargo)
            self._available_points.extend([
                Point(cargo.x + cargo.length, cargo.y, cargo.z),
                Point(cargo.x, cargo.y + cargo.width, cargo.z),
                Point(cargo.x, cargo.y, cargo.z + cargo.height)
            ])
            self._sort_available_points()
        return flag
    def _adjust_setting_cargo(self, cargo: Cargo):
        site = cargo.point
        temp = deepcopy(cargo)
        if not self.is_encasable(site, cargo):
            return None
```

```
        xyz = [site.x, site.y, site.z]
        # 序列化坐标以执行遍历递减操作，减少冗余
        for i in range(3): # 012 分别表示 xyz
            is_continue = True
            while xyz[i] > 1 and is_continue:
                xyz[i] -= 1
                temp.point = Point(xyz[0], xyz[1], xyz[2])
                for setted_cargo in self._setted_cargos:
                    if not _is_cargos_collide(setted_cargo, temp):
                        continue
                    xyz[i] += 1
                    is_continue = False
                    break
        cargo.point = Point(xyz[0], xyz[1], xyz[2]) # 反序列化
    def yanzhan_adjust_setting_cargo(self, cargo: Cargo):
        site = cargo.point
        temp = deepcopy(cargo)
        if not self.yanzhanis_encasable(site, cargo):
            return None
        xyz = [site.x, site.y, site.z]
        # 序列化坐标以执行遍历递减操作，减少冗余
        for i in range(3): # 012 分别表示 xyz
            is_continue = True
            while xyz[i] > 1 and is_continue:
                xyz[i] -= 1
                temp.point = Point(xyz[0], xyz[1], xyz[2])
                for setted_cargo in self._setted_cargos:
                    if not _is_cargos_collide(setted_cargo, temp):
                        continue
                    xyz[i] += 1
                    is_continue = False
                    break
        cargo.point = Point(xyz[0], xyz[1], xyz[2]) # 反序列化
    def daizi_adjust_setting_cargo(self, cargo: Cargo):
        site = cargo.point
        temp = deepcopy(cargo)
        if not self.daiziis_encasable(site, cargo):
            return None
        xyz = [site.x, site.y, site.z]
        # 序列化坐标以执行遍历递减操作，减少冗余
        for i in range(3): # 012 分别表示 xyz
            is_continue = True
```

```
            while xyz[i] > 1 and is_continue:
                xyz[i] -= 1
                temp.point = Point(xyz[0], xyz[1], xyz[2])
                for setted_cargo in self._setted_cargos:
                    if not _is_cargos_collide(setted_cargo, temp):
                        continue
                    xyz[i] += 1
                    is_continue = False
                    break
        cargo.point = Point(xyz[0], xyz[1], xyz[2]) #  反序列化
    def yanzhandaizi_adjust_setting_cargo(self, cargo: Cargo):
        site = cargo.point
        temp = deepcopy(cargo)
        if not self.yanzhandaiziis_encasable(site, cargo):
            return None
        xyz = [site.x, site.y, site.z]
        # 序列化坐标以执行遍历递减操作，减少冗余
        for i in range(3): # 012  分别表示  xyz
            is_continue = True
            while xyz[i] > 1 and is_continue:
                xyz[i] -= 1
                temp.point = Point(xyz[0], xyz[1], xyz[2])
                for setted_cargo in self._setted_cargos:
                    if not _is_cargos_collide(setted_cargo, temp):
                        continue
                    xyz[i] += 1
                    is_continue = False
                    break
        cargo.point = Point(xyz[0], xyz[1], xyz[2]) #  反序列化
    def save_encasement_as_file(self):
        file = open(f"{int(time())}_encasement.csv",'w',encoding='utf-8')
        file.write(f"index,x,y,z,length,width,height\n")
        i = 1
        for cargo in self._setted_cargos:
            file.write(f"{i},{cargo.x},{cargo.y},{cargo.z},")
            file.write(f"{cargo.length},{cargo.width},{cargo.height}\n")
            i += 1
        file.write(f"container,,,,{self}\n")
        file.close()

    def get_setted_cargos(self):
        return self._setted_cargos
```

```
    @property
    def length(self) -> int:
        return self._length

    @property
    def width(self) -> int:
        return self._width

    @property
    def height(self) -> int:
        return self._height

    @property
    def volume(self) -> int:
        return self.height * self.length * self.width


def _is_rectangles_overlap(rec1:tuple, rec2:tuple) -> bool:
    return not (
        rec1[0] >= rec2[2] or rec1[1] >= rec2[3] or
        rec2[0] >= rec1[2] or rec2[1] >= rec1[3]
    )


def _is_cargos_collide(cargo0: Cargo, cargo1: Cargo) -> bool:
    return (
        _is_rectangles_overlap(cargo0.get_shadow_of("xy"), cargo1.get_shadow_of("xy")) and
        _is_rectangles_overlap(cargo0.get_shadow_of("yz"), cargo1.get_shadow_of("yz")) and
        _is_rectangles_overlap(cargo0.get_shadow_of(
            "xz"), cargo1.get_shadow_of("xz"))
    )
from matplotlib import pyplot as plt     # 导入 Matplotlib 库中的 pyplot 模块
from matplotlib.figure import Figure     # 导入 Matplotlib 库中的 Figure 模块
import numpy as np                       # 导入 Numpy 库
from Encase3D._cargo import *            # 从 Encase3D 库中导入_Cargo 模块
from Encase3D._container import *        # 从 Encase3D 库中导入_Container 模块

plt.rcParams['axes.unicode_minus'] = False       # 配置 Matplotlib 绘图时的中文显示
```

```
plt.rcParams['font.sans-serif'] = ['SimHei']     # 配置 Matplotlib 绘图时的中文显示

fig:Figure = plt.figure()          # 创建一个 Figure 对象
ax = fig.add_subplot(1, 1, 1, projection='3d')     # 在 Figure 对象中添加一个 3D 坐标系

# 设置 3D 坐标系的视角
ax.view_init(elev=20, azim=40)

# 调整子图之间的间距
plt.subplots_adjust(top=1, bottom=0, right=1, left=0, hspace=0, wspace=0)

def draw_reslut(setted_container:Container):
    """
    绘制货物在集装箱中的摆放图
    """
    # 根据集装箱长、宽、高设置画布的坐标轴比例
    plt.gca().set_box_aspect((
        setted_container.length,
        setted_container.width,
        setted_container.height+200
    ))

    # 绘制集装箱的外框
    _draw_container(setted_container)

    # 遍历每个已放置的货物对象，绘制每个货物
    for cargo in setted_container._setted_cargos:
        _draw_cargo(cargo)

    # 展示绘制结果
    plt.show()

def _draw_container(container:Container):
    """
    绘制集装箱的外框
    """
    # 绘制长方体
    _plot_linear_cube(
        0,0,0,
        container.length,
        container.width,
        container.height
```

```
    )

def _draw_cargo(cargo:Cargo):
    """
    绘制货物
    """
    # 绘制立方体货物内部
    _plot_opaque_cube(
        cargo.x, cargo.y, cargo.z,
        cargo.length, cargo.width, cargo.height
    )

def _plot_opaque_cube(x=10, y=20, z=30, dx=40, dy=50, dz=60):
    """
    绘制表面不透明的立方体
    """
    xx = np.linspace(x, x+dx, 2)
    yy = np.linspace(y, y+dy, 2)
    zz = np.linspace(z, z+dz, 2)
    xx2, yy2 = np.meshgrid(xx, yy)
    ax.plot_surface(xx2, yy2, np.full_like(xx2, z))
    ax.plot_surface(xx2, yy2, np.full_like(xx2, z+dz))
    yy2, zz2 = np.meshgrid(yy, zz)
    ax.plot_surface(np.full_like(yy2, x), yy2, zz2)
    ax.plot_surface(np.full_like(yy2, x+dx), yy2, zz2)
    xx2, zz2= np.meshgrid(xx, zz)
    ax.plot_surface(xx2, np.full_like(yy2, y), zz2)
    ax.plot_surface(xx2, np.full_like(yy2, y+dy), zz2)

def _plot_linear_cube(x, y, z, dx, dy, dz, color='red'):
    """
    绘制表面不透明的长方体
    """
    xx = [x, x, x+dx, x+dx, x]
    yy = [y, y+dy, y+dy, y, y]
    kwargs = {'alpha': 1, 'color': color}
    ax.plot3D(xx, yy, [z]*5, **kwargs)
    ax.plot3D(xx, yy, [z+dz]*5, **kwargs)
    ax.plot3D([x, x], [y, y], [z, z+dz], **kwargs)
    ax.plot3D([x, x], [y+dy, y+dy], [z, z+dz], **kwargs)
    ax.plot3D([x+dx, x+dx], [y+dy, y+dy], [z, z+dz], **kwargs)
    ax.plot3D([x+dx, x+dx], [y, y], [z, z+dz], **kwargs)
```

```
import json
import pandas as pd
from Encase3D import *
from Encase3D import drawer
if __name__ == "__main__":
    with open('orderxi_data.json', 'r', encoding='utf-8') as f:
        data = json.load(f)
    cargos = [Cargo(210,200,30) for _ in range(1)]#210,200,30
    dingdan = 1
    case = []
    Cases = [[165, 120, 55], [200, 140, 70], [200, 150, 150], [270, 200, 90], [300, 200, 170]]
    A = []
    aa = [f'订单号：1']
    count = 0
    num = [0, 0, 0, 0, 0]

    for record in data[1:4]:#1:7955
        if  record['case（订单）'] == dingdan:
            cargos.extend([Cargo(record['l（长）'], record['w（宽）'], record['h（高）']) for _ in
range(record['num（数量）'])])
        else:
            aa = [f'订单号：{dingdan}"]
            dingdan_cargos = cargos
            num = [0, 0, 0, 0, 0]
            while(True):
                maxcargos = dingdan_cargos
                if maxcargos == []:
                    break
                else:
                    max = 0
                    for i in range(5):
                        case.append(Container(Cases[i][0], Cases[i][1], Cases[i][2]))
                        cargos1=failed_encase_cargos_into_container(dingdan_cargos,
case[count], VolumeGreedyStrategy)
                        count = count +1
                        # if cargos1 == []:
                        #     break
                        # elif len(cargos1) == len(cargos):
                        #     num = 1000
```

```
                                #         break
                                # else:
                                #         cargos = cargos1
                                #         num = num + 1
                                if max < len(cargos)-len(cargos1):
                                    maxi = i
                                    maxcargos = cargos1
                            num[maxi] = num[maxi]+1
                            dingdan_cargos = maxcargos
                    for i in range(5):
                        aa.append(str(num[0]))
                    A.append(aa)
                    print(num)
                    dingdan= record['case（订单）']
                    cargos = [Cargo(record['l（长）'], record['w（宽）'], record['h（高）']) for _ in
range(record['num（数量）'])]
        df = pd.DataFrame(A, columns=['订单号', '普通 1 号自营纸箱', '普通 2 号自营纸箱', '普通 3 号自
营纸箱','普通 4 号自营纸箱', '普通 5 号自营纸箱'])
        df.to_excel('result.xlsx', index=False, header=True)

问题一_袋子：
import pandas as pd
Vs = [47500,75000,132000,189000]#47500  75000     132000    189000

def process_excel_file(file_path):
    # 读取 Excel 文件
    df = pd.read_excel(file_path)
    # 从第二行开始遍历每一行
    for index in range(0, len(df)):
        # 选取当前行中除第一列以外的所有非空值，并转换为数字
        row = df.iloc[index, 1:]
        print(row)
        row = row[row.notna() & (row != '')]
        print(row)
        values = [int(v) for v in row]
        print(values)
        if len(values) == 0:
            # 如果该行中所有单元格都为空，则最小值和最小值位置均为 None
            min_value, min_index = None, None
        else:
            # 否则找到最小的值及其位置
            min_value = min(values)
```

```
                print(min_value )
                min_index = values.index(min_value) + 1
                print(min_index)

            # 在最后两列插入数据
            df.loc[index, '袋子数'] = min_value
            df.loc[index, '袋子编号'] = min_index
            df.loc[index, '订单耗材体积'] = min_value*Vs[min_index-1]
    # 保存修改后的 Excel 文件
    df.to_excel('T1 仅袋子求解结果.xlsx', index=False)

process_excel_file('daizi1result.xlsx')


问题一_混合
import pandas as pd
Vs = [47500,75000,132000,189000,1089000,1960000,4500000,4860000,10200000]#47500    75000
    132000    189000

def process_excel_file(file_path):
    # 读取 Excel 文件
    df = pd.read_excel(file_path)
    # 从第二行开始遍历每一行
    for index in range(0, len(df)):
        # 选取当前行中除第一列以外的所有非空值，并转换为数字
        row = df.iloc[index, 1:]
        print(row)
        row = row[row.notna() & (row != '')]
        print(row)
        values = [int(v) for v in row]
        print(values)
        if len(values) == 0:
            # 如果该行中所有单元格都为空，则最小值和最小值位置均为 None
            min_value, min_index = None, None
        else:
            # 否则找到最小的值及其位置
            min_value = min(values)
            print(min_value )
            min_index = values.index(min_value) + 1
            print(min_index)

        # 在最后两列插入数据
```

```
            df.loc[index, '箱子/袋子数'] = min_value
            df.loc[index, '总编号'] = min_index
            df.loc[index, '订单耗材体积'] = min_value*Vs[min_index-1]
    # 保存修改后的 Excel 文件
    df.to_excel('T1 混合求解结果.xlsx', index=False)

process_excel_file('hunheresult.xlsx')
```

问题二_箱子：

```
import json
import pandas as pd
from Encase3D import *
from Encase3D import drawer

V_max = [9080328, 28249524, 82133800, 207000000, 374467500]
if __name__ == "__main__":
    with open('orderxi_data.json', 'r', encoding='utf-8') as f:
        data = json.load(f)
    # num = 25837
    # cargos = [Cargo(170, 110, 27) for _ in range(7)]
    # for record in data[1:7955]:
    #         cargos.extend([Cargo(record['l（长）'], record['w（宽）'], record['h（高）']) for _ in
range(record['num（数量）'])])
    cargos = [Cargo(210, 200, 30) for _ in range(1)]    # 210,200,30
    dingdan = 1
    case = []
    cargos_set = {}

    Cases = [[165, 120, 55], [200, 140, 70], [200, 150, 150], [270, 200, 90], [300, 200, 170]]
    # for i in range(5):
    #         case.append(Container(Cases[i][0], Cases[i][1], Cases[i][2]))
    A = []
    aa = [f'订单号：1']
    count = 0
    for record in data[1:7955]:    # 1:7955
        cargos_set[record['case（订单）']] = list()
    for record in data[1:7955]:    # 1:7955
        if record['case（订单）'] == dingdan:
            # print(Cargo(record['l（长）'], record['w（宽）'], record['h（高）']))
            cargos_set[dingdan].append([record['l(长)'], record['w(宽)'], record['h(高)'],record['num
（数量）']])
```

```
                # cargos.extend(
                #           [Cargo(record['l（长）'], record['w（宽）'], record['h（高）']) for _ in
range(record['num（数量）'])])
            else:
                aa = [f"订单号：{dingdan}"]
                # for i in range(5):


                # num = 1
                #
                # while(True):
                #
                #       #print("订单：")
                #       #print(dingdan)
                #       # print(
                #       #
                #       # )
                #       case.append(Container(Cases[i][0], Cases[i][1], Cases[i][2]))
                #                   cargos1=failed_encase_cargos_into_container(cargos,    case[count],
VolumeGreedyStrategy)
                #       count = count + 1
                #       #print(cargos)
                #       # cargos1 = []
                #       # ####################
                #       # for c in range(len(cargos)):
                #       #       print(cargos[c].is_setted())
                #       #       if cargos[c].is_setted() == False:
                #       #             cargos1.append(cargos[c])
                #       if cargos1 == []:
                #             break
                #       elif len(cargos1) == len(cargos):
                #             num = 1000
                #             break
                #       else:
                #             cargos = cargos1
                #             num = num + 1
                # aa.append(str(num))
                # ####################
                A.append(aa)
                print(aa)
                dingdan = record['case（订单）']
                # cargos = [Cargo(record['l（长）'], record['w（宽）'], record['h（高）']) for _ in
range(record['num（数量）'])]
```

```
            cargos_set[dingdan].append([record['l(长)'], record['w(宽)'], record['h(高)'],record['num
(数量)']])

            if dingdan ==4040:
                t=1

    dingdan_fenlei = [set() for i in range(5)]

    # 读取 Excel 文件并将该文件的第一张工作表加载到 DataFrame 中。
    df = pd.read_excel('订单_总体积.xlsx')
    # 访问特定列
    data_dingdan = df['case']
    data_label = df['label']
    for i in range(len(data_dingdan)):
        dingdan_fenlei[data_label[i]].add(data_dingdan[i])
    V_min = [999999999 for i in range(5)]
    H_min = [0 for i in range(5)]
    W_min = [0 for i in range(5)]
    L_min = [0 for i in range(5)]
    for i in range(0,1):
        # while V_max[i] < V_max[i + 1]:
            for L in range(415,426,2):
                for W in range(190,201,2):
                    for H in range(300,311,2):
                        flag = True
                        if L * H * W < V_max[i] or L * H * W > 5 * V_max[i]:
                            continue
                        for item in dingdan_fenlei[i]:
                            case = Container(L, W, H)
                            cargos = []
                            for j in range(len(cargos_set[item])):
                                cargos.append(Cargo(cargos_set[item][j][0],
cargos_set[item][j][1], cargos_set[item][j][2]))
                            cargos1 = failed_encase_cargos_into_container(cargos, case,
VolumeGreedyStrategy)
                            if cargos1:
                                flag = False
                                break
                        if flag and L * H * W < V_min[i]:
                            V_min[i] = L * H * W
                            L_min[i] = L
                            W_min[i] = W
```

```
                                    H_min[i] = H
    print(V_max)
    print(V_min)
    print(L_min)
    print(W_min)
    print(H_min)

    # df = pd.DataFrame(A, columns=['订单号', '普通 1 号自营纸箱', '普通 2 号自营纸箱', '普通 3 号自营纸箱', '普通 4 号自营纸箱',
    #                                 '普通 5 号自营纸箱'])
    # df.to_excel('result.xlsx', index=False, header=True)
    for j in range(len(cargos_set[4040])):
        cargos.append(Cargo(cargos_set[4040][j][0], cargos_set[4040][j][1], cargos_set[4040][j][2]))
    print(failed_encase_cargos_into_container(cargos, Container(900,570,730), VolumeGreedyStrategy))
    # if failed_encase_cargos_into_container(cargos_set[4849], Container(500, 130, 140), VolumeGreedyStrategy):
    #     print('ture')
```

问题二_袋子：

```
import json
import pandas as pd
from Encase3D import *
from Encase3D import drawer

V_max = [11520000, 34679168, 98086630, 374467500]
if __name__ == "__main__":
    with open('orderdaizixi_data.json', 'r', encoding='utf-8') as f:
        data = json.load(f)
    # num = 25837
    # cargos = [Cargo(170, 110, 27) for _ in range(7)]
    # for record in data[1:7955]:
    #     cargos.extend([Cargo(record['l（长）'], record['w（宽）'], record['h（高）']) for _ in range(record['num（数量）'])])
    cargos = [Cargo(210, 200, 30) for _ in range(1)]    # 210,200,30
    dingdan = 1
    case = []
    cargos_set = {}

    Cases = [[165, 120, 55], [200, 140, 70], [200, 150, 150], [270, 200, 90], [300, 200, 170]]
    # for i in range(5):
    #     case.append(Container(Cases[i][0], Cases[i][1], Cases[i][2]))
```

```
    A = []
    aa = [f'订单号：1']
    count = 0
    for record in data[1:7955]:  # 1:7955
        cargos_set[record['case（订单）']] = list()
    for record in data[1:7955]:  # 1:7955
        if record['case（订单）'] == dingdan:
            # print(Cargo(record['l（长）'], record['w（宽）'], record['h（高）']))
            cargos_set[dingdan].append([record['l(长)'], record['w(宽)'], record['h(高)'],record['num（数量）']])
            # cargos.extend(
            #         [Cargo(record['l（长）'], record['w（宽）'], record['h（高）']) for _ in range(record['num（数量）'])])
        else:
            aa = [f'订单号：{dingdan}']
            # for i in range(5):

            # num = 1
            #
            # while(True):
            #
            #     #print("订单：")
            #     #print(dingdan)
            #     # print(
            #     #
            #     # )
            #     case.append(Container(Cases[i][0], Cases[i][1], Cases[i][2]))
            #                 cargos1=failed_encase_cargos_into_container(cargos,  case[count], VolumeGreedyStrategy)
            #     count = count + 1
            #     #print(cargos)
            #     # cargos1 = []
            #     # ####################
            #     # for c in range(len(cargos)):
            #     #     print(cargos[c].is_setted())
            #     #     if cargos[c].is_setted() == False:
            #     #         cargos1.append(cargos[c])
            #     if cargos1 == []:
            #         break
            #     elif len(cargos1) == len(cargos):
            #         num = 1000
            #         break
```

```
                #         else:
                #             cargos = cargos1
                #             num = num + 1
                # aa.append(str(num))
                # ####################
                A.append(aa)
                print(aa)
                dingdan = record['case（订单）']
                # cargos = [Cargo(record['l（长）'], record['w（宽）'], record['h（高）']) for _ in range(record['num（数量）'])]
                cargos_set[dingdan].append([record['l(长)'], record['w(宽)'], record['h(高)'],record['num（数量）']])


    dingdan_fenlei = [set() for i in range(4)]

    # 读取 Excel 文件并将该文件的第一张工作表加载到 DataFrame 中。
    df = pd.read_excel('订单_总体积.xlsx')
    # 访问特定列
    data_dingdan = df['case']
    data_label = df['label']
    for i in range(len(data_dingdan)):
        if data_dingdan[i] >=50 :
            continue
        dingdan_fenlei[data_label[i]].add(data_dingdan[i])
    V_min = [999999999 for i in range(4)]
    H_min = [0 for i in range(4)]
    W_min = [0 for i in range(4)]
    L_min = [0 for i in range(4)]
    for i in range(0,4):
        # while V_max[i] < V_max[i + 1]:
            for L in range(500,501,20):
                for W in range(500,501,20):
                    for H in range(500,1000,20):
                        flag = True
                        if L * H * W < V_max[i] :
                            continue
                        for item in dingdan_fenlei[i]:
                            case = Container(L, W, H)
                            cargos = []
                            for j in range(len(cargos_set[item])):
                                cargos.append(Cargo(cargos_set[item][j][0],
```

```
cargos_set[item][j][1], cargos_set[item][j][2]))
                                    cargos1 = daizifailed_encase_cargos_into_container(cargos, case,
VolumeGreedyStrategy)
                                    if cargos1:
                                        flag = False
                                        break
                                    else:
                                        print("true")
                                if flag and L * H * W < V_min[i]:
                                    V_min[i] = L * H * W
                                    L_min[i] = L
                                    W_min[i] = W
                                    H_min[i] = H
        print(V_max)
        print(V_min)
        print(L_min)
        print(W_min)
        print(H_min)

        # df = pd.DataFrame(A, columns=['订单号', '普通 1 号自营纸箱', '普通 2 号自营纸箱', '普通 3 号
自营纸箱', '普通 4 号自营纸箱',
        #                                '普通 5 号自营纸箱'])
        # df.to_excel('result.xlsx', index=False, header=True)
        for j in range(len(cargos_set[4040])):
            cargos.append(Cargo(cargos_set[4040][j][0], cargos_set[4040][j][1], cargos_set[4040][j][2]))
        print(failed_encase_cargos_into_container(cargos,                Container(900,570,730),
VolumeGreedyStrategy))
        #    if    failed_encase_cargos_into_container(cargos_set[4849],    Container(500,    130,    140),
VolumeGreedyStrategy):
        #        print('ture')



问题三_箱子：
import pandas as pd
#Vs = [1089000,1960000,4500000,4860000,10200000]#

def process_excel_file(file_path):
    # 读取 Excel 文件
    df = pd.read_excel(file_path)
    # 从第二行开始遍历每一行
    for index in range(0, len(df)):
        # 选取当前行中除第一列以外的所有非空值，并转换为数字
```

```
            row = df.iloc[index, 1:]
            print(row)
            row = row[row.notna() & (row != '')]
            print(row)
            values = [int(v) for v in row]
            print(values)
            if len(values) == 0:
                # 如果该行中所有单元格都为空，则最小值和最小值位置均为 None
                min_value, min_index = None, None
            else:
                # 否则找到最小的值及其位置
                min_value = min(values)
                print(min_value )
                min_index = values.index(min_value) + 1
                print(min_index)
                vvv = values[min_index]

            # 在最后两列插入数据
            df.loc[index, '箱子/袋子数'] = min_value
            df.loc[index, '总编号'] = (min_index+1)/2
            df.loc[index, '订单耗材体积'] = vvv
        # 保存修改后的 Excel 文件
        df.to_excel('T33 混合求解结果.xlsx', index=False)

process_excel_file('hunheresultT3.xlsx')
```