参赛编号:YRDMCM202300475

选题: \_A\_ (A或B或C) 参赛赛道: \_研究生\_（本科生、专科生或研究生）

**2023年第三届长三角高校数学建模竞赛**

题 目 启发式与贪心思想融合的装箱问题求解

摘 要：

随着中国快递市场的迅猛发展，2022年中国快递包裹数量已超过1000亿件，占据全球快递事务量一半以上。然而，在庞大且复杂的快递物流体系中，包裹打包环节至关重要。选取合适的包装耗材，特别是纸箱，对整个物流效率和成本控制具有重大意义。因此，本文旨在建立一个有效的数学模型，解决快递包裹装箱优化问题，提高装箱效率和经济效益。

本文简单分析了一种袋装类型耗材的简化近似，将复杂的形变问题转为简单的长方体，同时还保留了一定的近似度。对于问题一及其后续处理的基础，本文提出了一种基于贪心思想的先打分-后放置算法，并设计了一种简化物品放置的方法——凹角法，使之能够清晰简洁地处理物品的放置问题，将复杂多变的放置位选择问题简化为仅需对三个凹角的选择。先打分-后放置算法依赖于打分的权重系数，经验调整使人力不从心，于是结合启发式算法——差分进化来调整权重系数，取得了一定进步。对于第二问，我们对9种耗材的尺寸属性建模，先后用了粒子群优化和差分进化方法，求解出了一组尺寸属性。对于第三问，我们假设在装载时，物品与耗材总是处在最大形变状态，然后进行求解。

本文也存在许多有待进一步研究的不足，比如袋装类型耗材的简化近似模型在使用时并不能发挥出很好的效果，需要进一步理解柔性材料的放置操作，又比如打分的7个权重设置可能还可以再添加一些新的权重类型，提高放置的收益度。

关键词：贪心算法；装箱优化；差分进化；粒子群优化；多面体膨胀问题

**目录**

目录

[一、 问题重述 1](#_Toc7886)

[1.1 问题的背景 1](#_Toc13528)

[1.2 问题的提出 1](#_Toc27623)

[二、 基本假设及符号说明 2](#_Toc2212)

[2.1基本假设 2](#_Toc9312)

[2.2符号说明 2](#_Toc5191)

[三、 总体技术路线图 2](#_Toc28783)

[四、 数据预处理 3](#_Toc14544)

[4.1格式简化与数据剔除 3](#_Toc26581)

[4.2 可视化直观分析与数据排序 4](#_Toc9789)

[五、 问题一的求解 5](#_Toc7374)

[5.1进一步假设与符号说明 5](#_Toc10028)

[5.2模型建立 6](#_Toc29581)

[5.3模型求解 6](#_Toc27371)

[六、 问题二的求解 9](#_Toc21122)

[6.1模型建立 9](#_Toc18709)

[6.2求解思路 9](#_Toc23045)

[6.3粒子群优化 9](#_Toc4904)

[6.4求解结果 10](#_Toc1559)

[七、 问题三的求解 10](#_Toc27573)

[7.1问题分析 10](#_Toc29519)

[7.2求解思路 10](#_Toc6164)

[八、 模型评价与展望 12](#_Toc2109)

1. 问题重述
   1. 问题的背景

2022年，中国一年的快递包裹数量已经超过了1000亿件，占据了全球快递事务量的一半以上。令人惊讶的是，近几年来，中国每年新增的包裹数量已经相当于美国整个国家一年的包裹数量。而仅仅十年前，中国还是全球物流成本最高的国家。

在这个庞大且复杂的快递物流体系中，包裹打包环节是非常重要的一环。选取合适的包装耗材，尤其是合适的纸箱，对于整个物流效率和成本控制具有重要意义。考虑到中国的包裹基数之大，即使每个包裹的耗材成本只能略微降低，也可能带来极大的经济效益。

然而，如何有效地选取和使用纸箱，以实现包裹装箱的优化，如何根据包裹的大小和形状，以及可能的堆叠方式，选择最合适的纸箱，是一个值得深入研究的问题。这个问题也涉及到三维空间的**装箱问题**，需要考虑如何将各个包裹最优地放入一个给定的空间，以实现空间的最大化利用。

因此，对于中国这样一个全球最大的快递市场来说，研究和解决快递包裹装箱优化问题具有极其重要的实际意义和经济价值。本文的目标就是建立一个有效的数学模型，来解决这个实际问题，并尽可能提高包裹装箱的效率和经济效益。

* 1. 问题的提出

问题一:订单装箱优化

对于附件1中给出的订单数据和耗材数据，我们需要设计出一个有效的装载方案，使得每个订单都可以被适当地装入箱子或袋子中。设计方案的目标是使得使用的耗材数量最少，如果耗材数量相同，那么应优先选择总体积更小的耗材。我们需要提供每种耗材的使用总数以及总体积。

问题二:耗材尺寸优化

在问题一的基础上，我们现在需要针对附件1的数据，对耗材的尺寸进行优化。优化的目标是在保持耗材种类数量不变的情况下，通过改变耗材的尺寸，尽可能减少问题一中成功装载的物品所使用的箱子或袋子的数量。注意，优化后的耗材总体积不能超过原方案的总体积。如果耗材数量相同，那么应优先选择总体积更小的耗材。我们需要提供优化后的每种耗材的具体尺寸、使用总数以及总体积。

问题三:考虑柔性的装箱与尺寸优化

在问题一和问题二的基础上，我们需要考虑货物和耗材的物理性质。假设货物和耗材不再是完全刚性的，而是存在一定的柔性或者可以被轻微挤压。在这种情况下，我们需要重新完成问题一和问题二。在考虑耗材伸展时，我们假设长、宽、高的变化幅度都不会超过原尺寸的5%。

提示点:

**装载方案**：我们需要为每个订单考虑三种装载方案——箱装（只使用箱子作为耗材）、袋装（只使用袋子作为耗材）以及混合装（同时使用箱子和袋子作为耗材）。

**物品尺寸**：物品的长、宽、高可以任意互换。

袋子装载标准：当使用袋子装载物品时，袋子需要满足以下两个条件才能装下物品：袋子的长度+高度≥物品的长度+高度；袋子的宽度+高度≥物品的宽度+高度。

**订单分类**：具有相同序号的case视为同一订单。同一订单的物品可以装在同一箱（袋）子里，而不同订单的物品必须装在不同的箱（袋）子里。

**物品装载**：如果耗材无法装下，则可以不考虑该物品。

1. 基本假设及符号说明

2.1基本假设

**统一单位假设**：附件中所有长度单位均为㎝。

**严格旋转假设**：在装箱过程中，如果物品需要旋转，则必须保证物品旋转后其六面依旧平行于容器的六面。

**无厚度假设**：任何耗材本身都视为无厚度，即其体积等于其容积。

**无重叠假设**：任何两个物品或物品耗材之间都不能有重叠，即物品必须完全被耗材包含，不能超出容器的边界。

**无间隙假设**：物品与耗材之间，以及物品与物品之间在装载时没有任何间隙，即物品的边长完全可加，即如果两个物品在某一维度（如高度、长度或宽度）紧密堆叠，那么这两个物品的总边长等于各自边长的和，没有任何误差。

**无运动假设**：物品在装箱完成后就保持固定，不再改变自身位置。

**无分割假设**：物品不能被分割，每个物品必须作为一个整体装载。

2.2符号说明

|  |  |
| --- | --- |
| 符号 | 意义 |
| , | 容器,耗材 |
|  | 物品,商品,产品 |
|  | 耗材或物品的最长边,次长边,最短边,体积 |
| , | 订单与订单集合 |
|  | 容器集合 |
|  | 放置方案的集合与放置方案 |
|  | 耗材的尺寸规格 |

1. 总体技术路线图

数据预处理

问题一:订单装箱优化

问题二:耗材尺寸优化

机理分析

装箱函数设计与优化

算法优选

粒子群优化

贪心算法

差分进化

问题三:加入柔性考虑

差分进化

简化假设

1. 数据预处理

4.1格式简化与数据剔除

为了描述方便，下文中的“**耗材”与“容器”，“物品”与“商品”或“产品”指代相同。**

在本题中，由于订单数据中的物品尺寸差异较大，直接分析可能会存在一定的困难。因此，为了数据的一致性，我们首先对物品的长、宽、高进行了排序处理。分别得到了代表物品**最长边、次长边和最短边**的三个新变量，分别记为。

我们也对耗材数据也做了相同的处理,获得了相应的最长边、次长边和最短边，记为*,,*。我们还对耗材数据中每种耗材的名称与种类进行了数字映射，使之方便说明和运算。

由于袋装容器具有不规则形变的特性，分析物品装载方案可能具有一定的难度，下面我们将尝试用简单几何体以近似替代袋装容器。

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | | 5 |

**图1**

由文献[5]知，袋装容器的最大容积近似计算公式为：

,

其在最大容积下的形状如(图1.1)所示。

于是我们可以考虑在袋装容器达到最大膨胀状态下寻找一个内接的长方体，这个内接长方体的容积应尽量接近袋装容器的最大容积。

但这个方案的实现有一定难度，于是我们进一步简化，找到了一个更容易计算的膨胀状态[1]，其形状如（图1.2）所示。

其设计图（图1.3）非常简单，将正方形（矩形）的外接圆（椭圆）内翻，与图形本身所交的曲线作为弯折的线,将矩形的部分区域弯折，弯折时将弯折面垂直于原来矩形的面。

实际上在文献[8]中，也是通过这个方法（图1.4）开始近似逼近作证明。

由（图1.5）知，以该矩形为最大内接矩形的椭圆其长短轴分别是

则容器的最大高度（最短边）为  
此时容器的长宽（最长边与次长边）分别为

于是我们得到了袋装容器的近似替代的具有最大容积的长方体，而袋装容器的形状是可以随装入的**首个物品**的尺寸而调整的（只要物品的尺寸在允许的范围内），比如当物品的分别小于时，容器的最长边为

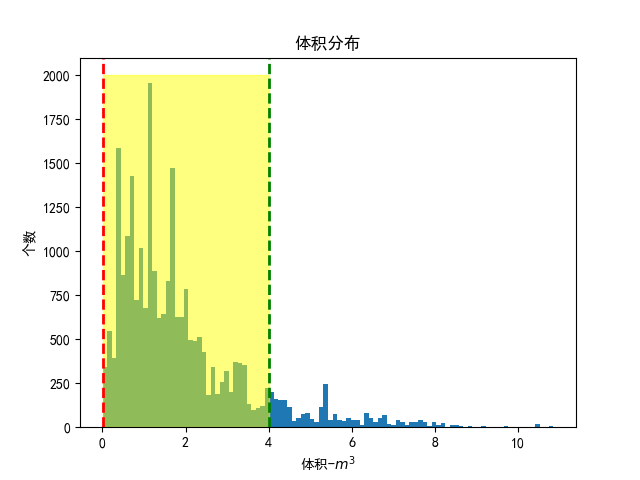
可知此时袋装容器的容积为

综上，我们先求得了袋装容器的近似替代长方体，同时它又是一个本身尺寸会随装入的首个物品的尺寸变化而变化着的长方体，因此该长方体可以被视为一种函数。

我们还进行了物品的筛选，对于三边均大于所有种类耗材对应三边的物品进行了剔除，这样的预处理步骤有助于简化后续的数据分析和处理。

4.2 可视化直观分析与数据排序

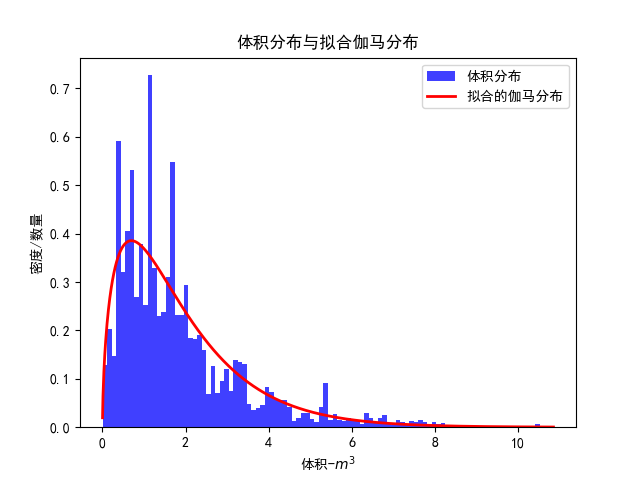
我们绘制了物品体积的分布图：



**图2**

经计算，可知从0到4.01m³的体积区间内，也就是图2的黄色区域内，汇聚了占总体90%的物品。

同时由图3可知，数据可能近似服从伽马分布形式的长尾分布：



**图3**

这这些统计特征为我们的后续处理提供了重要的信息，即**分治处理90%和10%的物品**。

于是我们进一步对数据依次按照订单编号（升序）、最长边（降序），次长边（降序），最短边（降序）进行了多级排序，即优先处理10%的大型物品。

最终我们将处理好的数据保存在了各类csv文件中，以方便调用。

1. 问题一的求解

5.1进一步假设与符号说明

在基本假设的基础上，添加如下说明：

假设此时数据预处理已经完成，订单数据集与耗材数据集均为按预定方式过滤、修改、排序过的二维表格。

**假设箱装容器以最短边,次长边作为底面，最长边作为高进行放置，装载物品时，总是从上部装入容器。**

**假设装载物品时,物品初始阶段以作为底面。**

假设袋装容器的体积等于内含物品的体积。

设订单，其中为中的订单编号，为订单中的物品个数，表示中的第个物品，其最长边，次长边，最短边如前所述为依次为,其体积为。有需要时会写成用于表明他从属于的订单。

设容器其中,对应了本题中具体的9种容器(耗材)的名称,当时为袋装容器,否则为箱装容器。

对于箱装容器，其最长边，次长边，最短边如前所述为依次为,其体积为。

对于袋装容器，其尺寸由首个装入的物品决定。

设全部可能方案数量为,可能的总装载方案为,*,*其中为第个订单的装载方案，有需要时会写成。

,其中是第个装载方案的第个种容器，在不致混淆的情况下简写成，，表示中的第 个物品,以及其摆放的规则, 在不致混淆的情况下,也简写成

表示中的元素个数。

5.2模型建立

由问题一的描述可知，这是一个多目标优化问题。具体来说，这个问题具有以下两个优化目标：

1.最小化所需容器的数量；

2.在所需容器数量相同的情况下，最小化容器的总体积。

于是我们设计目标函数为:

若对有则求:

5.3模型求解

由于不同订单之间的物品不可混装于同一容器，我们可专注于一个订单所对应的装载方案内部，下面对的分析可从拓展到任意装载方案上。

5.4机理分析

为了尽可能减少所需容器的数量，我们应该想到尽可能提高容器的利用率。因此我们想到了用贪心的思路设计算法。

贪心算法是一种解决优化问题的方法，它在每一步都选择当前看起来最优的选择，希望这样的局部最优选择最终能导致全局最优解。贪心算法的主要优点是简单、易于实现，且在某些问题上能够有效地找到最优解。

在本文中，我们的装箱算法的主要思想是尝试将待装载物品依次放入已选容器和新容器中，通过评估不同放置方式的评分来决定物品的放置。

以下是算法的简要概述：

初始化一个空列表作为已选容器列表。

从待装载物品序列中取出新物品。

初始化一个空列表作为评分信息表。

如果已选容器列表非空，尝试将新物品放入已选容器，并根据放置结果为每种情况打分。同时，尝试将新物品放入新容器，同样根据放置结果打分。

如果已选容器列表为空，强制添加第一个元素。

选择最优评分方案。

将新物品插入已选容器或新容器，并更新容器的**装载角**表。

**装载角**是指一种物品放置的策略，即每次都选择**凹角（也称凹面）**处放置物品，这样能极大地简化放置操作。

其主要目标是找到适合物品“整齐”放置的位置，“整齐”是指尽可能与之前放置的物品形成公共边，减少因不规则放置导致的空间浪费。

如图4.1至4.4所示，每次往凹角放置一个物品，都会填补1个凹角，顺带又产生1到3个凹角。当放置的物品与之前的物品具有公共边时，则认为放置得非常“整齐”如图4.2所示，此时并不会让凹角增多，我们认为这样的放置策略是优秀的，因为“整齐”的摆放有助于空间的利用。

同时我们只考虑正面，右面和上面的凹角，对于图4.5的情况是不会出现的，因为我们已经排序过订单数据，下一个出来的物品其规格总是小于之前的物品的。

此外，对于凹面的处理需要考虑合并计算，如图4.6所示，此时画面左侧已经堆叠了两个物品,他们因为具有公共边,从而使得两个凹角被合并为了一个凹角，此时我们就应该在代码上识别出这种状况而修正它。

|  |  |  |
| --- | --- | --- |
| 1  20230514_140847000_iOS | 2  20230514_140937000_iOS | 3  20230514_140921000_iOS |
| 4  20230514_140901000_iOS | 5  20230514_141018000_iOS | 6  20230514_150630000_iOS |

**图4**

5.5装箱函数的设计

经过上述分析可知，装箱函数设计的难点有两个，其一是**对凹角的操作**，需要实现生成与合并凹角，其二是**评分函数**的设计需要对每个凹角上的放置提供合理的评分，以方便选出最优的放置位置。

对于凹角的操作，经过实验探究，我们发现可以用如下的思路来实现：

创建新装载角：首先，创建三个新的装载角：装载角x、装载角y和装载角z。每个新装载角的坐标和尺寸都与当前装载角、物品三维尺寸有关。

初始化不插入装载角列表：创建一个空列表，用于存储可能不需要插入的新装载角。

遍历新装载角：从新创建的装载角x、装载角y和装载角z中，逐个取出新装载角并进行判断。

如果新装载角的尺寸中存在0，将新装载角添加到不插入装载角列表中。

遍历容器的装载角表，取出待比较的装载角。如果待比较装载角不是对应类型的，跳过此次循环。

计算待选坐标，包括：待比较装载角的坐标加上其尺寸的x、y和z分量。

如果待选坐标包含新装载角的坐标，那么更新待比较装载角的尺寸。根据新装载角的类型，将新装载角的尺寸x、y和z分量分别添加到待比较装载角的尺寸中。

结束循环：当所有新创建的装载角都被遍历和处理后，算法结束。

对于评分函数，我们直觉上实行了如下的评分设计：

边占比：根据物品边长和装载角边长计算边占比。如果需要考虑边因素，计算x、y、z方向上的边占比，并取最大值。

面积占比：根据物品的面积和装载角的面积计算占比。如果需要考虑面积因素，计算三组面积占比，分别为x-y、x-z、y-z方向的面积占比，并取最大值。

体积占比：根据物品的体积和装载角的体积计算占比。如果需要考虑体积因素，计算物品体积与装载角体积的占比。

总体积占比：根据物品的体积和容器的总体积计算占比。如果容器信息的第二个元素为1，计算物品体积与容器总体积的占比。

返回评分：根据以上计算的边占比、面积占比、体积占比和总体积占比，乘以新旧权重并取最大值作为评分返回。

由于整个装箱函数只有一个大循环,以及对以保存容器进行评分的循环,因此其复杂度是，

我们用python实现了装箱函数，并执行了任务，以下是得到的结果：

|  |  |  |  |
| --- | --- | --- | --- |
| 物品数量 | 仅箱型耗材数量 | 仅袋型耗材数量 | 混合数量(箱+袋) |
| 24768 | 14111 | 23341 | 16274=12871+3403 |
|  | 仅箱型耗材体积 | 仅袋型耗材体积 | 混合体积(箱+袋)m³ |
|  | 74631 | 46457 | 85522=71288+14233 |

5.5装箱函数的优化

由于种种问题和原因，我们实现的贪心算法成绩并不算理想,尤其是在仅使用袋装耗材的情况下效果非常不好，究其原因, 除了我们使用了不同于本题原先的袋装模型外, 我们还认为其核心在于评分的权重参数的调整。

5.6差分进化算法

我们采用了差分进化算法来调整我们的权重参数。

差分进化（Differential Evolution, DE）算法是一种优化算法，主要用于求解多维连续空间中的全局最优解。该算法是一种进化算法，它利用种群的进化策略来逐步改进解的质量。与其他进化算法（如遗传算法）相比，差分进化算法在实现简单、收敛速度快以及调参要求低等方面具有优势。

在本问题中使用差分进化是因为其收敛速度较快，而且适合对实数进行优化，而权重参数恰恰是实数。

我们将七个权重参数直接编码为基因序列，边界范围均设为[0,5]，采用随迭代递减的变异率和交叉率的参数来提高搜索速率和收敛速度。

结果分析

最终，我们用差分进化算法经过一段时间的参数调整，得到了如下的结果：

对应的参数分别是:

仅袋型: [0.,0.78404743,1.72001252,4.07501294,5.0.,0.92733388]

仅箱型: [0.78763425,5.,2.61205767,5,3.63201236,2.6547866,0.4176828 ]

混合型:[0.54901561,4.80096797,2.352933,5.,0.,5.,0.12909913]

分别对应了7种评分标准的权重:新容器,旧容器,边占比,面积占比,体积占比,总体积占比,优先贴底

|  |  |  |  |
| --- | --- | --- | --- |
| 物品数量 | 仅箱型耗材数量 | 仅袋型耗材数量 | 混合数量(箱+袋) |
| 24768 | 12678 | 22498 | 14836=11433+3403 |
|  | 仅箱型耗材体积 | 仅袋型耗材体积 | 混合体积(箱+袋)m³ |
|  | 70956 | 46457 | 81824=71288+14234 |

1. 问题二的求解

6.1模型建立

根据问题描述，我们需要在保持耗材种类数量不变的情况下，通过调整耗材尺寸，尽可能减少装载物品所需的箱子或袋子数量。同时，优化后的耗材总体积不能超过原方案的总体积。这是一个多目标优化问题，因为我们需要在满足多个目标（如最小化箱子或袋子数量、不超过原方案总体积等）的情况下寻找最优解。

设容器规格为，不致混淆时可写成 ，

均为正实数，故可用中的一点来代表,于是就有 来表示采用了某一规格的容器. 设为上一问中求出的装载方案。

于是我们可以设计目标函数为：

若对有则求:

6.2求解思路

在本题中，我们将继续采用问题一中所设计的装载函数，并结合差分进化算法得到的优化评分参数作为求解基础。进一步地，我们计划使用粒子群优化（Particle Swarm Optimization, PSO）算法对耗材规格进行优化。

6.3粒子群优化

PSO（Particle Swarm Optimization）是一种基于群体智能的优化算法，旨在通过模拟生物体群体行为来解决优化问题。PSO算法最初由Kennedy和Eberhart于1995年提出，它是一种基于种群的随机优化算法，通过模拟鸟群或鱼群等自然界中的集体行为来搜索解空间中的最优解。

在PSO算法中，解空间中的每个解被看作是一个粒子，每个粒子都有一个位置和速度，它们通过不断地更新自己的速度和位置来搜索最优解。每个粒子的位置和速度根据当前的位置和速度以及群体中历史最优位置和全局最优位置来更新。通过这种方式，粒子可以在解空间中寻找到最优解。

PSO算法的优点在于它能够快速找到全局最优解，并且具有较强的鲁棒性。

为了表示 9 种耗材及其每种耗材的 3 维数据，我们使用一个形状为 (9, 3) 的二维数组。在这个数组中，每一行表示一种耗材，每一列表示一个维度（长、宽、高）。

假设每种耗材的长、宽和高的范围分别为 [1, 400]、[1, 400] 和 [1,400]。

我们使用体积与数量之比最大化作为适应度函数，旨在让算法尽可能减少耗材数量而增大体积。

由于这是一个27维的优化问题，维数较高，粒子群求解有一定的时间要求，因此我们也辅以差分进化算法同时执行，可作为比较。在这里差分进化的基因编码为27维数据，对应了9种耗材的三维长宽高，使用截断的方式获取到整数解其适应度函数为耗材使用数量与体积之比，对于大于上一题得到的结果的数据采用惩罚函数的机制来规避。

6.4求解结果

|  |  |  |  |
| --- | --- | --- | --- |
| 耗材规格 | 总体使用量 | 体积 | 算法 |
| [[普通1号袋,袋,213,1,400]  [普通2号袋,袋,386,254,188]  [普通3号袋,袋,177,343,185]  [普通4号袋,袋,239,301,22]  [普通1号自营纸箱,箱,43,118,398]  [普通2号自营纸箱,箱,133,25,174]  [普通3号自营纸箱,箱,191,163,58]  [普通4号自营纸箱,箱,96,180,126]  [普通5号自营纸箱,箱,178,221,228]]  使用第二问的权重系数 | 10361 | 68715 | 差分进化 |
|  | 运行失败 | 运行失败 | 粒子群优化 |

1. 问题三的求解

7.1问题分析

在问题一和问题二的基础上，我们需要考虑货物和耗材的物理性质约束。现假设货物和耗材具有一定程度的柔性，允许轻微挤压。在此约束条件下，重新完成问题一和问题二的求解。

7.2求解思路

为了简化问题，我们假设在装载时，物品与耗材总是处在最大形变状态，即物品的边长总是缩小5%，而耗材的边长总是增加5%。

7.3求解结果

第一问

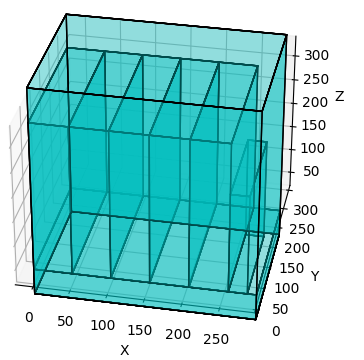
|  |  |  |  |
| --- | --- | --- | --- |
| 物品数量 | 仅箱型耗材数量 | 仅袋型耗材数量 | 混合数量(箱+袋) |
| 24768 | 12380 | 19976 | 13191=11736+1455 |
|  | 仅箱型耗材体积 | 仅袋型耗材体积 | 混合体积(箱+袋)m³ |
|  | 67003 | 39873 | 72134=65048+7085 |

、

第二问

|  |  |  |  |
| --- | --- | --- | --- |
| 耗材规格 | 使用量 | 体积 | 算法 |
| [[普通1号袋,袋,313,109,400]  [普通2号袋,袋,1,202,290]  [普通3号袋,袋,214,42,131]  [普通4号袋,袋,208,391,349]  [普通1号自营纸箱,箱,149,400,156]  [普通2号自营纸箱,箱,23,36,1]  [普通3号自营纸箱,箱,1,194,1]  [普通4号自营纸箱,箱,1,74,177]  [普通5号自营纸箱,箱,34,1,358]] | 8755 | 79489 | 差分进化 |

效果图：



1. 模型评价与展望

在文章的最后，我们简单做个总结回顾。

我们在文章中提出了一种简化袋装耗材模型的简单方法，但对其进一步研究缺乏足够的经验，使之不能很好地配合我们的工作，导致容积浪费较多。同时也提出了一种有迹可循的三维放置物品的方法——**凹角法**，将放置物品的问题简化成了判断凹角是否值得摆放的问题。

我们的模型求解基于贪心+启发式优化，存在很多可改进的地方：

比如，权重参数的设定，虽然目前设置了7种权重，但还是欠缺了点，比如我们可以考虑迭代次数作为一个变量。

又比如，袋装耗材的利用率还可以进一步提高，可以进一步增加是否可装载判断的灵活性。

以上提到的几点都可以作为日后进一步研究的方向。

参考文献

[1]http://origami.oschene.com/cp/pure-tbp-box.pdf

[2][https://math.stackexchange.com/questions/2855975/what-is-the-[3]maximum-volume-that-can-be-contained-by-a-sheet-of-paper](https://math.stackexchange.com/questions/2855975/what-is-the-maximum-volume-that-can-be-contained-by-a-sheet-of-paper)

[4]https://www.ics.uci.edu/~eppstein/junkyard/teabag.html

[5]<https://en.wikipedia.org/wiki/Paper_bag_problem>

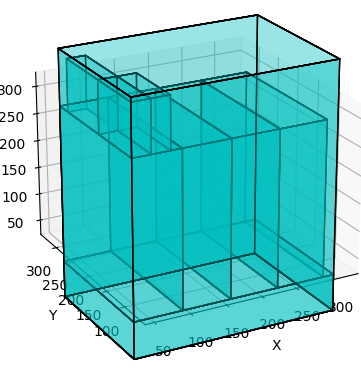
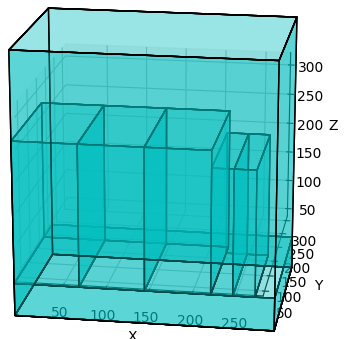
[6]http://origami.oschene.com/archives/2008/04/06/design-ideas-up-the-yin-yang/

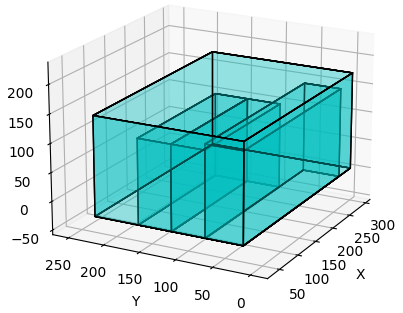
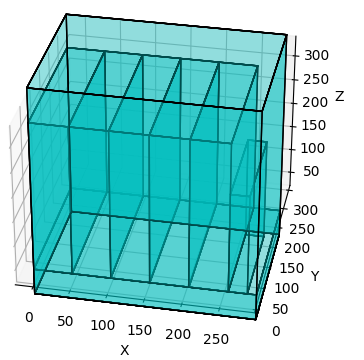
[7]IGOR PAK- inflating polyhedral surfaces(2006)

[8]IGOR PAK AND JEAN-MARC SCHLENKER - PROFILES OF INFLATED SURFACES(2009)

附录

部分装箱效果图





装箱函数代码

*class* 权重:  
 新容器 = 1  
 旧容器 = 2  
 边占比 = 2  
 面积占比 = 1  
 体积占比 = 1  
 总体积占比 = 1  
 优先贴底 = 1  
  
 *def \_\_init\_\_*(*self*,权重值表=*None*):  
 *self*.权重值表 = [*self*.新容器,*self*.旧容器,*self*.边占比,*self*.面积占比,*self*.体积占比,*self*.总体积占比,*self*.优先贴底]  
  
 *if* 权重值表 *is not None*:  
 *self*.新容器= 权重值表[0]  
 *self*.旧容器= 权重值表[1]  
 *self*.边占比= 权重值表[2]  
 *self*.面积占比= 权重值表[3]  
 *self*.体积占比= 权重值表[4]  
 *self*.总体积占比= 权重值表[5]  
 *self*.优先贴底 = 权重值表[6]  
 *# for 键,值 in enumerate(权重值表):  
 # self.权重值表[键]=值  
  
  
def* 获取权重():  
 *return  
  
def* 获取容器(容器表,容器编号):  
 *for* 容器 *in* 容器表:  
 *if* 容器[0]==容器编号:  
 *return* 容器  
  
  
*def* 袋装容器\_获取最大立方体的三边(容器数据):  
 h = 容器数据[2] + 容器数据[4]  
 w = 容器数据[3] + 容器数据[4]  
 最长边 = (2 - math.sqrt(2)) \* h  
 次长边 = (2 - math.sqrt(2)) \* w  
 最短边 = (math.sqrt(2) - 1) \* h  
 *return* 最长边, 次长边, 最短边  
 *pass  
  
def* 袋装容器\_获取最长边(容器数据, 长, 宽):  
 h = 容器数据[2] + 容器数据[4]  
 w = 容器数据[3] + 容器数据[4]  
 *return* h - 长  
  
*def* 袋装容器\_根据首个装入物返回尺寸(已装载容器):  
 容器信息 = 已装载容器[0]  
 长,宽,高 = 已装载容器[1][0][1]  
 最长边 = 袋装容器\_获取最长边(容器信息,长,宽)  
 *return* 长,宽,最长边  
  
*def* 物品操作\_旋转(物品信息, 底部信息):  
 *if* 底部信息 == 0:  
 *return* 物品信息[1], 物品信息[2], 物品信息[3] *# ab  
 elif* 底部信息 == 1:  
 *return* 物品信息[1], 物品信息[3], 物品信息[2] *# ac  
 elif* 底部信息 == 2:  
 *return* 物品信息[2], 物品信息[3], 物品信息[1] *# bc  
 elif* 底部信息 == 3:  
 *return* 物品信息[2], 物品信息[1], 物品信息[3] *# ba  
 elif* 底部信息 == 4:  
 *return* 物品信息[3], 物品信息[1], 物品信息[2] *# ca  
 elif* 底部信息 == 5:  
 *return* 物品信息[3], 物品信息[2], 物品信息[1] *# cb  
 else*:  
 *raise ValueError*("错误的底部信息!")  
  
*def* 装载函数操作\_装载角能装下(装载角, 新物品):  
 装载角尺寸 = -np.sort(-np.array(装载角[1]))  
 新物品尺寸 = np.array(新物品[1:])  
 *return* np.all(装载角尺寸 >= 新物品尺寸)  
  
*def* 装载函数操作\_评分(容器信息,x边, y边, z边, 角x边, 角y边, 角z边,新旧权重=权重.旧容器,需要面积=*True*,需要边=*True*,需要体积=*True*,权重实例=*None*):  
 权:权重 = 权重实例 *if* 权重实例 *else* 权重  
 *return* 新旧权重 \* *max*(  
 权.边占比 \* *max*(x边 / 角x边, y边 / 角y边, z边 / 角z边) *if* 需要边 *else* 0 ,  
 权.面积占比 \* *max*(  
 (权.优先贴底\*x边 \* y边) / (角x边 \* 角y边),  
 (x边 \* z边) / (角x边 \* 角z边),  
 (y边 \* z边) / (角y边 \* 角z边)) *if* 需要面积 *else* 0,  
 权.体积占比 \* (x边 \* y边 \* z边 )/ (角x边 \* 角y边 \* 角z边) *if* 需要体积 *else* 0,  
 权.总体积占比 \*(x边 \* y边 \* z边 )/(容器信息[2]\*容器信息[3]\*容器信息[4]) *if* 容器信息[1]==1 *else* 0  
 *#权.总体积占比 \*(x边 \* y边 \* z边) / (角x边 \* 角y边 \* 角z边)* )  
  
*def* 装载函数操作\_新增装载角(容器,旧装载角,物品三维):  
 装载角x =[辅助.坐标加(旧装载角[0], [物品三维[0], 0, 0]),  
 [旧装载角[1][0]-物品三维[0],物品三维[1],物品三维[2]], 0]  
  
 装载角y = [辅助.坐标加(旧装载角[0], [0, 物品三维[1], 0]),  
 [物品三维[0],旧装载角[1][1]-物品三维[1],物品三维[2]], 1]  
  
 装载角z = [辅助.坐标加(旧装载角[0], [0, 0, 物品三维[2]]),  
 [物品三维[0],物品三维[1],旧装载角[1][2]-物品三维[2]], 2]  
 不插入装载角列表=[]  
  
 *for* 新装载角 *in* [装载角x,装载角y,装载角z]:  
 *if* 0 *in* 新装载角[1]:  
 不插入装载角列表.append(新装载角)  
 *for* 待比较装载角 *in* 容器[2]:  
 待选坐标 = [  
 辅助.坐标加(待比较装载角[0],[待比较装载角[1][0],0,0]),  
 辅助.坐标加(待比较装载角[0], [ 0,待比较装载角[1][1], 0]),  
 辅助.坐标加(待比较装载角[0], [0, 0, 待比较装载角[1][2]])  
 ]  
 *if* 新装载角[0] *in* 待选坐标:  
 新增尺寸 = [  
 新装载角[1][0] *if* 新装载角[2] == 0 *else* 0,  
 新装载角[1][1] *if* 新装载角[2] == 1 *else* 0,  
 新装载角[1][2] *if* 新装载角[2] == 2 *else* 0,  
 ]  
 待比较装载角[1] = 辅助.坐标加(待比较装载角[1],新增尺寸)  
 *for* 新装载角 *in* [装载角x, 装载角y, 装载角z]:  
 *if* 新装载角 *not in* 不插入装载角列表:  
 容器[2].append(新装载角)  
  
*def* 装载函数(待装载物品序列, 待选容器表,权重\_=*None*):  
 *"""  
 容器的数据格式: 容器元素=[物品编号,物品三维,插入坐标],容器=[本身属性[容器信息],容器元素表[容器元素],装载角表[装载角],容积],  
 装载角=[坐标(x,y,z),尺寸(l,h,w),类型(x=0,y=1,z=2)]  
 物品信息=[订单号,长,宽,高], 待装载物品序列 = List[物品信息]  
 容器信息=[名称(数字),类型(数字),长,宽,高], 容器种类表 = List[容器信息]  
 已选容器列表=[容器]  
 评分信息=[旋转,坐标,已选容器编号,新容器编号,得分]  
 解释:  
 评分信息是指一个物品将被如何放置到容器中,  
 旋转: 取值0,1,2,3,4,5分别表示长,宽,高三个数据的全排列取前两个作为物品的底面.  
 已选容器编号: 如果已选容器编号=-1则考虑创建新容器  
 新容器编号: 是指容器的名称编号, 通常为-1,只有需要新建容器时,才会等于指定的名称编号  
  
 """* 权重\_ = 权重() *if* 权重\_ *is None else* 权重\_  
  
 已装载容器序列:"" = [] *# 已装载容器序列[容器[]]  
 for* 新物品编号,新物品 *in enumerate*(待装载物品序列):  
 *# 评分环节* 评分表 = [] *# 包含物品评分与操作的表  
  
 if* 已装载容器序列:  
 *for* 容器编号, 容器 *in enumerate*(已装载容器序列):  
 装载角表 = 容器[2]  
 *for* 装载角 *in* 装载角表:*# 容器2 即装载角  
 if* 装载函数操作\_装载角能装下(装载角, 新物品):  
 角x边, 角y边, 角z边 = 装载角[1]  
 *for* i *in range*(6):  
 x边, y边, z边 = 物品操作\_旋转(新物品, i)  
 *if* 角x边 < x边 *or* 角y边 < y边 *or* 角z边 < z边:  
 *continue* 评分 = 装载函数操作\_评分(容器,x边, y边, z边,角x边, 角y边, 角z边,权重实例=权重\_)  
 评分表.append([i,装载角[0],容器编号,-1,评分])  
 *for* \_,待选容器 *in enumerate*(待选容器表):  
 待选容器编号=待选容器[0]  
 角x边, 角y边, 角z边 = 待选容器[2], 待选容器[3], 待选容器[4]  
 *if* 待选容器[1] == 1:  
 *for* i *in range*(6):  
 x边, y边, z边 = 物品操作\_旋转(新物品, i)  
 *if* 角x边<x边 *or* 角y边<y边 *or* 角z边<z边:  
 *continue* 评分 = 装载函数操作\_评分(待选容器,x边, y边, z边, 角x边, 角y边, 角z边,新旧权重=权重\_.新容器,权重实例=权重\_)  
 评分表.append([i, (0,0,0), -1, 待选容器编号, 评分])  
 *else*:  
 容器最长, 容器次长, 容器最短 = 袋装容器\_获取最大立方体的三边(待选容器)  
  
 *for* i *in range*(6):  
 x边, y边, z边 = 物品操作\_旋转(新物品, i)  
 *# 超过这些厚度和宽度就无法装载了, 所以直接跳过* 物品导出最长边 = 袋装容器\_获取最长边(待选容器, x边, y边)  
 *if not* (角x边+角z边>=x边+z边 *and* 角y边+角z边>=y边+z边 ) *or* 物品导出最长边 < z边:  
 *continue  
 else*:  
 评分 = 装载函数操作\_评分(待选容器, x边, y边, z边, x边, y边, 物品导出最长边, 新旧权重=权重\_.新容器,  
 需要边=*False*,需要面积=*False*,权重实例=权重\_  
 )  
 *# 评分 = 权重\_.新容器 \* (max(比例得分, 体积得分))* 评分表.append([i, (0,0,0), -1, 待选容器编号, 评分])  
 *pass  
  
 if len*(评分表)==0:  
 *for* \_,待选容器 *in enumerate*(待选容器表[0:4]):  
 *if* 新物品[1]+新物品[3]<=待选容器[2]+待选容器[4] *and* 新物品[2]+新物品[3]<=待选容器[3]+待选容器[4]:  
 容器 = [待选容器.copy(), [[新物品编号, 新物品[1:4], (0,0,0)]], [],新物品[1]\*新物品[2]\*新物品[3]]  
 已装载容器序列.append(容器)  
 *break  
 continue* 最佳操作 = *max*(评分表, key=*lambda* x: x[-1])  
 *if* 最佳操作[-1] == -1:  
 *raise ValueError*("没有最佳分数, 请检查问题")  
  
 旋转,坐标,容器编号,新容器编号,得分 = 最佳操作  
 x边, y边, z边 = 物品操作\_旋转(新物品, 旋转)  
 *# 植入环节  
 if* 容器编号!=-1: *# 此时将新物品插入旧的容器中* 容器 = 已装载容器序列[容器编号]  
 容器[1].append([新物品编号,(x边, y边, z边),坐标])  
 *if* 容器[0][1]==0:  
 容器[3]+= x边\*y边\*z边  
 装载角 = [角 *for* 角 *in* 容器[2] *if* 角[0]==坐标][0]  
 容器[2].remove(装载角)  
 装载函数操作\_新增装载角(容器,装载角,[x边, y边, z边])  
 *else*:*# 插入新容器* 容器 = [获取容器(待选容器表,新容器编号).copy(),[[新物品编号,(x边, y边, z边),坐标]],[],0]  
 容器[3] += x边\*y边\*z边 *if* 容器[0][1]==0 *else* 容器[0][2]\*容器[0][3]\*容器[0][4]  
 已装载容器序列.append(容器)  
 *if* 容器[0][1]==1:  
 装载角 = [[0,0,0],容器[0][2:5],2]  
 *else*:  
 装载角 = [[0, 0, 0],[x边, y边, 袋装容器\_获取最长边(容器[0], x边, y边)], 2]  
 装载函数操作\_新增装载角(容器,装载角, [x边, y边, z边])  
 *pass  
 return* 已装载容器序列