武汉大学国家网络安全学院
课程报告

基于启发式搜索算法的三维装箱问题求解

专 业 名 称 ： 网络空间安全

课 程 名 称 ： 高级算法设计与分析

指 导 教 师 ： 林海

学 号 姓 名 ： 2022202210100 滕　龙

二 〇 二 二 年 十 二 月

目　录

第一部分 背景介绍

1.1 问题描述

经典的装箱问题的描述是:将一定数量的物品放入有一定容量的容器中，使得每个容器中的物品之和不超过容器容量并获得某种最佳的效益。待放入的物品和使用的箱子均为长方体，这是因为一般产品在生产结束后为了运输搬运的方便往往采用长方体包装，而运输容器一般也都是长方体的。

按照物品的种类数目可分为同构装箱问题和异构装箱问题。只有一种物品类型的装箱问题被称为同构装箱问题。有少数几种物品类型且每种类型数量较多的装箱问题被称为弱异构装箱问题。强异构装箱问题则有许多物品类型且每种类型数量较少。

除了以上提到的分类,针对特定需求的装箱问题有很多约束，例如重心约束，承重能力约束等。本文主要考虑的是单容器异构三维装箱问题，可以形式化定义如下:

给定一个容器(其体积为 V)和一系列待装载的物品，容器和物品的形状都是长方体。问题的目标是要确定一个可行的物品放置方案使得在满足给定装载约束的情况下，容器中包含的物品总体积 S 尽可能大，填充率尽可能高。填充率定义为：

$$\rho = \frac{S}{V} \times 100\%$$

可行的放置方案要求放置满足如下四个基本约束条件:

（1）被装载的物品必须完全被包含在容器中。

（2）任何两个被装载的物品不能互相重叠。

（3）所有被装载的物品以与容器平行的方式放置，即不能斜放。

（4）静态装箱：从 n 个物品中选取 m 个物品，并且实现 m 个物品在容器内的摆放。

除了以上的基本约束以外，针对实际问题，本方案还考虑以下五个额外的约束。

（1）稳定约束：每个被装载的物品必须得到容器底部或者其他物品的支撑，也就是说，禁止被装载的物品悬空。

（2）浮点尺寸：物品的尺寸最多考虑到小数点后两位。

（3）顺序摆放：物品在容器内摆放时，按照从内到外或者是从下到上的顺序摆放。

（4）摆放姿态：对于一个 L×W×H 的物品而言，其在容器中有 6 种不同的摆放姿态。

（5）实用性：装箱算法能够较快地实现，在 2 秒以内完成。

1.2 方案描述

目前对于三维装箱问题算法的研究，多集中在异构装箱问题上，因为在实际应用中，往往会涉及到种类繁多的物品。装箱问题是一个 NP-Hard 问题，在多项式时间复杂度内无法保证找到问题的最优解。采用传统的精确搜索技术来求解装箱问题时，往往有可能产生“组合爆炸”的现象，因此在实际求解时是不现实的，从而启发式求解算法成为了理论研究和实际应用的首选。

在本实现中，物品每一次只放置一个，没有回溯机制，一旦物品被放进容器之中，那么它就不会被重新拿出来。选择需要放进容器中的物品可以通过以下几种不同的形式逻辑来完成：

• First Fit：最先匹配，将未放置的物品放进第一个有足够空间的容器中，如果没有合适的容器，那就将未放置的物品放置到一个新的空容器中。

• First Fit Decreasing：递减最先匹配，与最先匹配几乎完全相同，只是物品在放置进容器之前，按照容器转载量递减的顺序进行排序。

• Last Fit：最末匹配，将未放置的物品放进最后一个有足够空间的容器中，于最先匹配非常类似，但是容器的顺序是相反的。如果没有合适的容器，那就将未放置的物品放进一个新的空容器中。

• Best Fit：最佳匹配，最佳匹配算法是将物品放置在所有容器中选取一个最为“合适”的容器来放置。

在本方案中采用的两种主要的启发式装箱算法分别是 First Fit Decreasing 和 Best Fit，这两者有较快的运行时间，并且运行结果与最优解也较为接近，空间的利用率也较高。

第二部分 算法介绍

2.1 三维最佳匹配算法

在将物品放置入容器的过程中，我们需要考虑三个方向上的扩展来放置，即 x 轴方向，y 轴方向和 z 轴方向。在放置物品之前，我们首先选择一个“轴点”，这个轴点是物体左侧后方下侧的顶点，这个轴点坐标作为物品在容器内位置的记录。

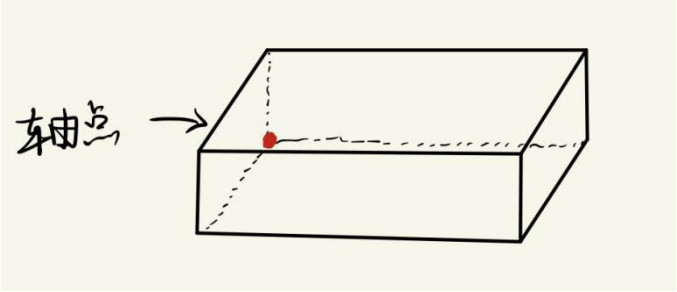

图 1：轴点示意图

如果当前的轴点位置无法被放置进入容器中的话，我们就将物体采取 6 个方式来旋转，直至物体能够被放置入容器中的当前轴点位置上。如果在尝试了 6 中不同旋转方式的情况下，物体依然无法放入，那么就先放入另一个物体，并将当前的物体添加到未放置列表中。在所有的物体放入完毕后，我们会尝试着放入未放置列表中的物体。在算法开始执行之前，一个空容器的首个轴点会被初始化为（0,0,0）。

2.1.1 算法流程简述

最佳匹配算法分为以下几个步骤：

（1）初始化轴点为（0,0,0），对所有要放入的物体按照体积大小进行排序，在放置过程中，优先放置体积大的物体。

（2）在第一个物体放置完毕后，之后每放置一个物体之前，首先生成一个新的轴点。这个轴点是按照上一个物体的轴点在依次按照 x 轴，y 轴，z 轴三个方向上生成的三个可放置轴点。

（3）在新的放置点上，尝试 6 种不同的物体旋转方式，直到物品可以被放进容器之中。由于并没有回溯机制，因此在方案中，物体按照一种方式放进了某个可放置轴点之中，就会再开始尝试放入下一个物体。

（4）如果尝试了 3 个可放置轴点，每个轴点上都尝试了 6 种不同的旋转方式，都无法将物品放入容器中，那么就将物品放入未放置列表中，跳过该物品处理下一个。

（5）重复 2-4 步，处理完所有的物品之后，所有可放置的物品已经被放入箱子中，同时也有一些物品保存在了未放置列表中，此时再来尝试处理这些未放置的物品。

2.1.2 算法伪代码

Algorithm 1: 3D Best Fit

```
1: procedure 3DBestFit(Bin, ItemList):
2:     fitted_item = []          #Record thoes fitted items.
3:     unfitted_item = []        #Record thoes unfitted items.
4:     ItemList.sort(reserve=True)   #Arrange items from largest to smallest.
5:     for item in ItemList:
6:         fiitted = False       #Fit tag.
7:         pivot = [0,0,0]       #The first pivot is (0,0,0)
8:         if Bin is empty:
```

```
9:      procedure Put_Item_in_Bin(Bin,Item,pivot):
10:         for item in Rotation(item):      #Try 6 different rotation
11:             if pivot[0]+item.length>Bin.length #if Bin cannot hold item
12:             or pivot[1]+item.width>Bin.width
13:             or pivot[2]+item.depth>Bin.depth:
14:                 continue
15:             if intersect(current_Bin_items,item) == False #If this item
16:                 fitted_item.append()    #isn't intersect with other items
17:                 item.position = pivot    #Update item's position
18:                 fitted = True
19:             else:
20:                 break
21:         if item not in fitted_item:
22:             unfitted_item.append()#Try 6 rotations item cannot put in Bin
23:             item.position = NULL #Don't update item's position
24:     elif Bin is not empty:
25:         for axis in range(3):#try to put item in three axises
26:             for ib in fitted_item: #get the newest pivot
27:                 if axis == 0: #x-axis
28:                     pivot=[ib.position[0]+item.length,
29:                            ib.position[1],
30:                            ib.position[2]]
31:                 if axis == 1: #y-axis
32:                     pivot=[ib.position[0],
33:                            ib.position[1]+item.width,
34:                            ib.position[2]]
35:                 if axis == 2: #z-axis
36:                     pivot=[ib.position[0],
37:                            ib.position[1],
```

```
38:                                    ib.position[2]+item.height]
39:                          Put_Item_in_Bin(Bin,Item,pivot)
40:     3DBestFit(Bin, unfitted_item)#handle those unfitted items
41:     return
```

2.1.3 物体旋转算法

在本方案中，我们定义了物体六种不同的旋转类型：

旋转类型	对应编号
L×W×H	0
W×L×H	1
W×H×L	2
H×W×L	3
H×L×W	4
L×H×W	5

在实现中，我们默认物品采用了 L×W×H 的形式，如果采用了不同的旋转类型，我们会对应修改物体的长宽高数据。在具体放置时，方案会尝试所有的旋转方案，直到物体能够被放置入容器中。

2.1.4 物体重合检测

在实现中，我们需要考虑一种情况就是容器内的两个物体，是否存在着重合这一类情况，这是违背实际的。因此我们需要在实现中检测是否存在物体重合的情况。

我们将长方体的物体重合检测转化成成三个不同的面重合检测问题，即 L×W 面，L×H 面以及 W×H 面。

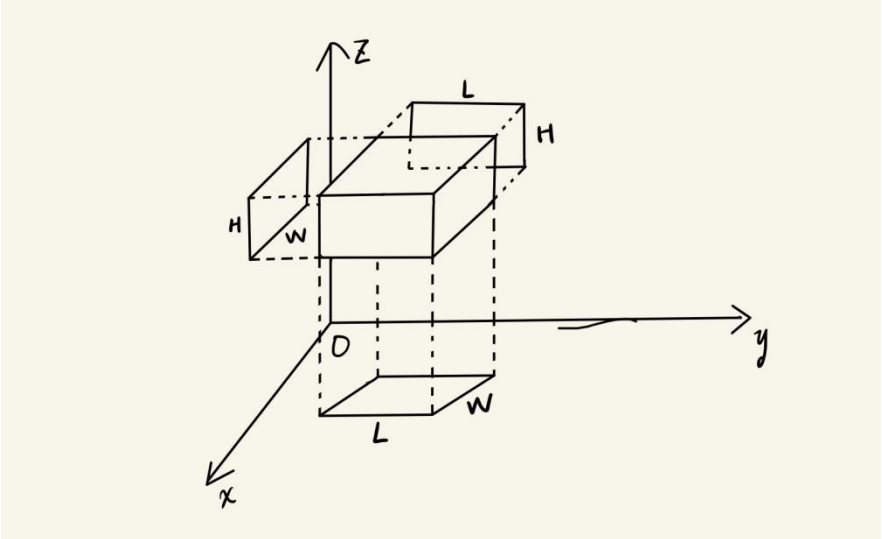

图 2：三个平面示意图

这样做的原理是：如果这长方体在 xoy 平面、xoz 平面和 yoz 平面的投影都没有重合，那么两个长方体也一定没有重合。

在判断两个物体是否在某一个平面上的投影有重合时，我们采用了如下步骤：

（1）根据两个长方体的轴点坐标和双方再这个平面上的边长数据，计算出两个长方体在这个平面上的投影矩形的中心。

（2）计算两个矩形中心在 x 轴和 y 轴上的距离，如果这两个矩形没有相交，那么这两者在 x 轴或 y 轴上的距离应大于双方对应边长之和的一半，反之如果两者在 x 轴和 y 轴上的距离应小于双方对应边长之和的一半，说明出现了重合的情况。

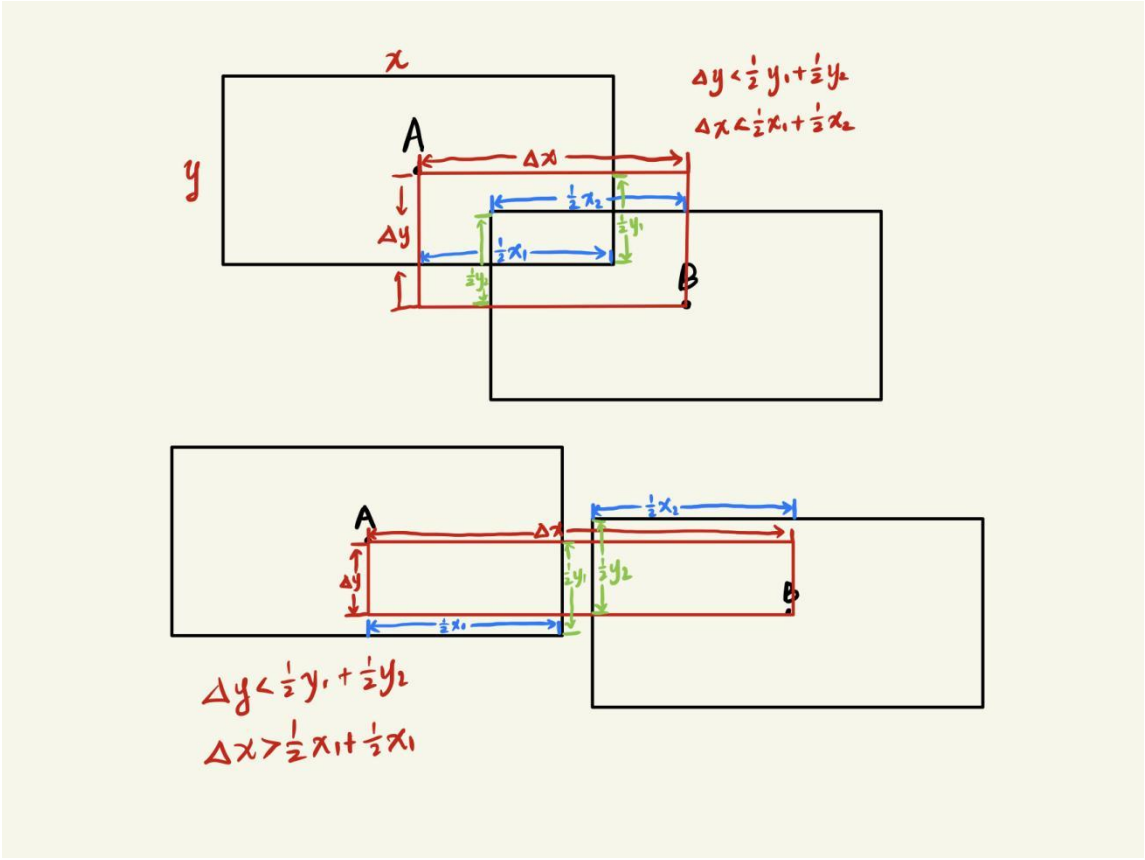

图 3：判定两个矩形是否重合

2.1.5 最坏情况下的运行时间

在上述算法中，存在以下五个循环：

（1）在所有的 ImtemList 中循环放置每个 item，如果存在 n 个 item 的情况下，这个循环会被执行 n 次。

（2）在有空间放入 item 的时候，item 需要和 Bin 中已有的物品判断是否存在重合的情况。如果一共存在 n 个物品时，则需要运行 n-1 次，进行 n-1 次判断。

（3）在所有已放置的物品上，来寻找下一个可供物品放置的轴点，在最坏的情况下，需要遍历完所有物品，也就是执行 n-1 次才能够找到一个合适的轴点。

（4）在新放入物品需要选择轴点时，会从 x 轴，y 轴和 z 轴选取 3 个方向来判断在上一个物品的哪一个方向来放置，会执行 3 次。

（5）在放入物品时，需要尝试 6 中不同的旋转形式，因此会执行 6 次。

因此，总的时间复杂度为：

$$O(3DBestFit) = n \times (n-1) \times (n-1) \times 3 \times 6 = O(n^3)$$

在最差的情况下，算法的时间复杂度为$O(n^3)$，这依然是一个多项式级别的时间，因此该方案的运行效率还是比较高的。

2.1.6 最好情况下的运行时间

最好情况下和最差情况相比，区别在于第三个循环中，每一次不需要遍历完所有的物体就能够找到一个可供下一个物体放置的轴点，也就是说每次上一个物体上都恰好有足够下一个物体放置的轴点。那么这时，此循环只用执行 1 次。那么总的时间复杂度为：

$$O(3DBestFit) = n \times (n-1) \times 1 \times 3 \times 6 = O(n^2)$$

因此在最好情况中，算法的时间复杂度为$O(n^2)$，此时算法的运行效率已经相当高了。

2.2 三维递减最先匹配算法

2.2.1 算法流程简述

三维递减优先匹配算法其他步骤与三维最佳匹配算法是相似的，只是在放置具体物体的过程中小有差异。具体的算法流程如下：

（1）初始化轴点为（0,0,0），对所有要放入的物体按照体积大小进行排序，在放置过程中，优先放置体积大的物体。

（2）在第一个物体放置完毕后，之后每放置一个物体之前，首先生成一个新的轴点。这个轴点是按照上一个物体的轴点在依次按照 x 轴，y 轴，z 轴三个方向上生成的三个可放置轴点。

（3）在放置下一个物体之前，首先将物体的最长边旋转到与容器的最长边平行的位置上，如果一个容器尺寸是 10×9×8，即长度大于宽度，宽度大于高度；此时有一个物品尺寸是 1×2×3，那么就将物品旋转为 3×2×1，再放置物体。也就是保持物体的最长边贴着容器的最长边放置。

（4）如果物品旋转成 3×2×1 无法放入容器，那就将物体旋转成 2×1×3 或者 2×3×1，也就是将第二长边对应箱子的长边来放置；如果还是无法放入的话，就将物体旋转成 1×3×2 或者 1×2×3，也就是将第三长边来对应箱子的长边来放置。

（5）如果在三个轴点上都尝试了 6 种不同的旋转方式，都无法将物品放入容器中，那么就将物品放入未放置列表中，跳过该物品处理下一个。

（6）重复 2-4 步，处理完所有的物品之后，所有可放置的物品已经被放入箱子中，同时也有一些物品保存在了未放置列表中，此时再来尝试处理这些未放置的物品。

2.2.2 算法伪代码

其实这个算法只是调整了物品在放入容器时的旋转的顺序，其他的步骤与最佳匹配算法是一样的，物体旋转的伪代码如下所示：

Algorithm 2: First Fit Decreasing Rotation

```
1: procedure FFDRotation(Item):
2:     RotationType=[LWH=0,WHL=1,WLH=2,HLW=3,HWL=4,LHW=5]
3:     for rt in RotationType:
4:         if rt == 0:
5:             Item.size = [Item.length,Item.width,Item.height]
6:         if rt == 1:
7:             Item.size = [Item.width,Item.height,Item.length]
8:         if rt == 2:
9:             Item.size = [Item.width,Item.length,Item.height]
10:        if rt == 3:
11:            Item.size = [Item.height,Item.length,Item.width]
12:        if rt == 4:
13:            Item.size = [Item.height,Item.width,Item.length]
14:        if rt == 5:
15:            Item.size = [Item.length,Item.height,Item.width]
16:        else:
17:            Item.size = []
18:        return Item
```

第三部分 方案测试

3.1 基础约束测试

在本节中，我们测试方案是否满足 4 个基础约束：即一、被装载的物品必须完全被包含在容器中；二、任何两个被装载的物品不能互相重叠；三、所有被装载的物品以与容器平行的方式放置，即不能斜放；四、从 n 个物品中选取 m 个

物品，并且实现 m 个物品在容器内的摆放。

测试用例为：

容器体积为 10×1×1，物品体积为 1×1×1，一共有 15 个同样的物品，如果方案满足这四个约束的话，那么方案能够在 15 个物品中选取 10 个并排放进容器中，此时的空间利用率应为 100%。

测试结果：

```
Bin(10.00 x 1.00 x 1.00 = 10.00)
FITTED ITEMS:
( 1 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [0, 0, 0]  rotation is 0
( 2 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [Decimal('1.00'), 0, 0]  rotation is 0
( 3 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [Decimal('2.00'), 0, 0]  rotation is 0
( 4 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [Decimal('3.00'), 0, 0]  rotation is 0
( 5 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [Decimal('4.00'), 0, 0]  rotation is 0
( 6 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [Decimal('5.00'), 0, 0]  rotation is 0
( 7 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [Decimal('6.00'), 0, 0]  rotation is 0
( 8 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [Decimal('7.00'), 0, 0]  rotation is 0
( 9 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [Decimal('8.00'), 0, 0]  rotation is 0
( 10 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [Decimal('9.00'), 0, 0]  rotation is 0
UNFITTED ITEMS:
( 1 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [0, 0, 0]  rotation is 5
( 2 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [0, 0, 0]  rotation is 5
( 3 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [0, 0, 0]  rotation is 5
( 4 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [0, 0, 0]  rotation is 5
( 5 ) BoxA(1.00 x 1.00 x 1.00 = 1.00) position is [0, 0, 0]  rotation is 5
utility rate of space is 100.00%
used  0.0020046234130859375 seconds
```

可以观察到，是个 1×1×1 的箱子被并排放进容器之中，同时他们的轴点坐标分别是（0,0,0）,...,（9,0,0），同时有 5 个物品未能放入容器之中，容器的总空间利用率为 100%。这说明方案在装箱的过程中，能够保证被装载的物品必须完全被包含在容器中；同时，任何两个被装载的物品不能互相重叠；其次，所有被装载的物品以与容器平行的方式放置；最后从 n 个物品中选取 m 个物品，并且实现 m 个物品在容器内的摆放。

3.2 浮点尺寸测试

在本节中，我们测试方案是否能够对尺寸精度为小数点后两位的物品进行装箱。在本方案中，我们采用了 Decimal 来进行精确的浮点数运算，而不是像浮点数据那样的一个不精确的表述。所有的输入数据都会首先被标准化为 Decimal 格式，来保证运算的正确性。

测试用例为：

物品名称	长度	宽度	高度	体积
Bin	19.68	13.75	3.37	911.92
BoxA	3.94	1.96	1.97	15.21

BoxB	3.94	1.96	1.97	15.21
BoxC	3.94	1.96	1.97	15.21
BoxD	7.87	3.94	1.97	61.09
BoxE	7.87	3.94	1.97	61.09
BoxF	7.87	3.94	1.97	61.09
BoxG	7.87	3.94	1.97	61.09
BoxH	7.87	3.94	1.97	61.09
BoxI	7.87	3.94	1.97	61.09

测试结果：

```
Bin(19.68 x 13.75 x 3.37 = 911.92)
FITTED ITEMS:
( 1 ) BoxD(7.87 x 3.94 x 1.97 = 61.09) position is [0, 0, 0]  rotation is 0
( 2 ) BoxE(7.87 x 3.94 x 1.97 = 61.09) position is [Decimal('7.87'), 0, 0]  rotation is 0
( 3 ) BoxF(7.87 x 3.94 x 1.97 = 61.09) position is [Decimal('15.74'), 0, 0]  rotation is 2
( 4 ) BoxG(7.87 x 3.94 x 1.97 = 61.09) position is [0, Decimal('3.94'), 0]  rotation is 0
( 5 ) BoxH(7.87 x 3.94 x 1.97 = 61.09) position is [Decimal('7.87'), Decimal('3.94'), 0]  rotation is 0
( 6 ) BoxI(7.87 x 3.94 x 1.97 = 61.09) position is [0, Decimal('7.88'), 0]  rotation is 0
( 7 ) BoxA(3.94 x 1.96 x 1.97 = 15.21) position is [Decimal('7.87'), Decimal('7.88'), 0]  rotation is 0
( 8 ) BoxB(3.94 x 1.96 x 1.97 = 15.21) position is [Decimal('11.81'), Decimal('7.88'), 0]  rotation is 0
( 9 ) BoxC(3.94 x 1.96 x 1.97 = 15.21) position is [Decimal('15.75'), Decimal('7.88'), 0]  rotation is 2
UNFITTED ITEMS:
utility rate of space is 45.20%
used  0.0 seconds
```

测试结果说明方案可以满足尺寸精度为 2 位小数点以内的装箱方案搜索。

3.3 稳定约束及顺序摆放测试

在本节中，我们测试方案是否能够满足稳定性约束，也就是说是否会存在物体悬空的情况，同时检测方案是否能够先放置从下往上放置物品。

测试用例为：

物品名称	长度	宽度	高度	体积
Bin	4	4	3	48
BoxA	3	3	1	9
BoxB	4	2	1	8
BoxC	1	1	1	1

测试结果为：

```
Bin(4.00 x 4.00 x 3.00 = 48.00)
FITTED ITEMS:
( 1 ) BoxA(3.00 x 3.00 x 1.00 = 9.00) position is [0, 0, 0]  rotation is 0
( 2 ) BoxB(4.00 x 2.00 x 1.00 = 8.00) position is [Decimal('3.00'), 0, 0]  rotation is 3
( 3 ) BoxC(1.00 x 1.00 x 1.00 = 1.00) position is [0, Decimal('3.00'), 0]  rotation is 0
UNFITTED ITEMS:
utility rate of space is 37.50%
used  0.0 seconds
```

我们可以观察到，方案将 BoxA 放在了起始点（0,0,0）的位置上，BoxB 旋转放在了 BoxA 的旁边，BoxC 放在了 BoxA 的上面。通过这个简单的例子，可以展现方案（1）从底层开始放置，底层全部摆满了再放上层，也就是从下往上顺序摆放；（2）不存在物品悬空的情况。这是由于在算法实现中，在一个新物品要放入容器的时候，优先在 x 轴和 y 轴的方向上进行扩展，在 x 轴和 y 轴上没有空间时，才会考虑 z 轴方向上的扩展。同时，在放置之前，我们首先会对 Box 进行排序，优先放大物品，再放小物品。

这样的结果就是在实现中，下层是体积大的物品，上层是体积小的物品，能够一定程度地避免悬空现象。

3.4 功能及效率测试

在本节中，我们采用了 3DLoad-test-dataSet 中的数据来对方案进行功能和效率测试，主要的测试目标是测试方案是否能够正确高效运行，测试标准是空间利用率和时间。

3.4.1 三种箱子

测试用例 E1-1：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	108	76	30	40
BoxB	110	43	25	33
BoxC	92	81	55	39

测试结果 E1-1：

```
utility rate of space is 80.46%
used 0.77 seconds
```

空间利用率为 80.46%，用时 0.77 秒。

测试用例 E1-2、E1-3：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	91	54	45	32
BoxB	105	77	72	24
BoxC	79	78	48	30

测试结果 E1-2、E1-3：

```
utility rate of space is 74.18%
used 0.40 seconds
```

空间利用率为 74.18%，用时 0.4 秒。

测试用例 E1-4：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	60	40	32	64
BoxB	98	75	55	40
BoxC	60	59	39	64

测试结果 E1-4：

```
utility rate of space is 79.18%
used 2.76 seconds
```

空间利用率为 79.18%，用时 2.76 秒。

测试用例 E1-5：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	78	37	27	63
BoxB	89	70	25	52
BoxC	90	84	41	55

测试结果 E1-5：

```
utility rate of space is 77.88%
used 1.52 seconds
```

空间利用率为 77.88%，用时 1.52 秒。

3.4.2 五种箱子

测试用例 E2-1：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	108	76	30	24
BoxB	110	43	25	7
BoxC	92	81	55	22
BoxD	81	33	28	13
BoxE	120	99	73	15

测试结果 E2-1：

```
utility rate of space is 82.23%
used 0.27 seconds
```

空间利用率为 82.23%，用时 0.27 秒。

测试用例 E2-2：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	49	25	21	22
BoxB	60	51	41	22
BoxC	103	76	64	28
BoxD	95	70	62	25
BoxE	111	49	26	17

测试结果 E2-2：

```
utility rate of space is 86.64%
used 0.59 seconds
```

空间利用率为 86.64%，用时 0.59 秒。

测试用例 E2-3：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	88	54	39	25
BoxB	94	54	36	27
BoxC	87	77	43	21
BoxD	100	80	72	20
BoxE	83	40	36	24

测试结果 E2-3：

```
utility rate of space is 77.96%
used 0.84 seconds
```

空间利用率为 77.96%，用时 0.84 秒。

测试用例 E2-4：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	90	70	63	16
BoxB	84	78	28	28
BoxC	94	85	39	20
BoxD	80	76	54	23
BoxE	69	50	45	31

测试结果 E2-4：

```
utility rate of space is 80.56%
used 1.03 seconds
```

空间利用率为 80.56%，用时 1.03 秒。

测试用例 E2-5：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	74	63	61	22
BoxB	71	60	25	12
BoxC	106	80	59	25
BoxD	109	76	42	24
BoxE	118	56	22	11

测试结果 E2-5：

```
utility rate of space is 78.88%
used 0.39 seconds
```

空间利用率为 78.88%，用时 0.39 秒。

3.4.3 八种箱子

测试用例 E3-1：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	108	76	30	24
BoxB	110	43	25	9
BoxC	92	81	55	8
BoxD	81	33	28	11
BoxE	120	99	73	11
BoxF	111	70	48	10
BoxG	98	72	46	12
BoxH	95	66	31	9

测试结果 E3-1：

```
utility rate of space is 77.82%
used 0.31 seconds
```

空间利用率为 77.82%，用时 0.31 秒。

测试用例 E3-2：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	97	81	27	10
BoxB	102	78	39	20
BoxC	113	46	36	18

BoxD	66	50	42	21
BoxE	101	30	26	16
BoxF	100	56	35	17
BoxG	91	50	40	22
BoxH	106	61	56	19

测试结果 E3-2：

```
utility rate of space is 78.78%
used 1.47 seconds
```

空间利用率为 78.78%，用时 1.47 秒。

测试用例 E3-3：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	88	54	39	16
BoxB	94	54	36	14
BoxC	87	77	43	20
BoxD	100	80	72	16
BoxE	83	40	36	6
BoxF	91	54	22	15
BoxG	109	58	54	17
BoxH	94	55	30	9

测试结果 E3-3：

```
utility rate of space is 83.84%
used 0.83 seconds
```

空间利用率为 83.84%，用时 0.83 秒。

测试用例 E3-4：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	49	25	21	16
BoxB	60	51	41	8
BoxC	103	76	64	16
BoxD	95	70	62	18
BoxE	111	49	26	18
BoxF	85	84	72	16
BoxG	48	36	31	17
BoxH	86	76	38	6

测试结果 E3-4：

```
utility rate of space is 86.83%
used 0.66 seconds
```

空间利用率为 86.83%，用时 0.66 秒。

测试用例 E3-5：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	113	92	33	23
BoxB	52	37	28	22
BoxC	57	33	29	26
BoxD	99	37	30	17
BoxE	92	64	33	23
BoxF	119	59	39	26
BoxG	54	52	49	18
BoxH	75	45	35	30

测试结果 E3-5：

```
utility rate of space is 83.60%
used 2.78 seconds
```

空间利用率为 83.60%，用时 2.78 秒。

3.4.4 十种箱子

测试用例 E4-1：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	49	25	21	13
BoxB	60	51	41	9
BoxC	103	76	64	11
BoxD	95	70	62	14
BoxE	111	49	26	13
BoxF	85	84	72	16
BoxG	48	36	31	12
BoxH	86	76	38	11
BoxI	71	48	47	16
BoxJ	90	43	33	8

测试结果 E4-1：

```
utility rate of space is 87.67%
used 0.79 seconds
```

空间利用率为 87.67%，用时 0.79 秒。

测试用例 E4-2：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	97	81	27	8
BoxB	102	78	39	16
BoxC	113	46	36	12
BoxD	66	50	42	12
BoxE	101	30	26	18
BoxF	100	56	35	13
BoxG	91	50	40	14
BoxH	106	61	56	17
BoxI	103	63	58	12
BoxJ	75	57	41	13

测试结果 E4-2：

```
utility rate of space is 78.82%
used 1.30 seconds
```

空间利用率为 78.82%，用时 1.30 秒。

测试用例 E4-3：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	86	84	45	18
BoxB	81	45	34	19
BoxC	70	54	37	13
BoxD	71	61	52	16
BoxE	78	73	40	10
BoxF	69	63	46	13
BoxG	72	67	56	10
BoxH	75	75	36	8
BoxI	94	88	50	12
BoxJ	65	51	50	13

测试结果 E4-3：

```
utility rate of space is 80.65%
used 1.45 seconds
```

空间利用率为 80.65%，用时 1.45 秒。

测试用例 E4-4：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	113	92	33	15
BoxB	52	37	28	17
BoxC	57	33	29	17
BoxD	99	37	30	19
BoxE	92	64	33	13
BoxF	119	59	39	19
BoxG	54	52	49	13
BoxH	75	45	35	21
BoxI	79	68	44	13
BoxJ	116	49	47	22

测试结果 E4-4：

```
utility rate of space is 84.29%
used 2.84 seconds
```

空间利用率为 84.29%，用时 2.84 秒。

测试用例 E4-5：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	118	79	51	16
BoxB	86	32	31	8
BoxC	64	58	52	14
BoxD	42	42	32	14
BoxE	64	55	43	16
BoxF	84	70	35	10
BoxG	76	57	36	14
BoxH	95	60	55	14
BoxI	80	66	52	14
BoxJ	109	73	23	18

测试结果 E4-5：

```
utility rate of space is 82.68%
used 1.06 seconds
```

空间利用率为 82.68%，用时 1.06 秒。

3.4.5 十五种箱子

测试用例 E5-1：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	93	73	44	6
BoxB	60	60	38	7
BoxC	105	73	60	10
BoxD	90	77	52	3
BoxE	66	58	24	5
BoxF	106	76	55	10
BoxG	55	44	36	12
BoxH	82	58	23	7
BoxI	74	61	58	6
BoxJ	81	39	24	8
BoxK	71	65	39	11
BoxL	105	97	47	4
BoxM	114	97	69	5
BoxN	103	78	55	6
BoxO	93	66	55	6

测试结果 E5-1：

```
utility rate of space is 81.21%
used 0.55 seconds
```

空间利用率为 81.21%，用时 0.55 秒。

测试用例 E5-2：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	108	76	30	12
BoxB	110	43	25	12
BoxC	92	81	55	6
BoxD	81	33	28	9
BoxE	120	99	73	5
BoxF	111	70	48	12
BoxG	98	72	46	9

BoxH	95	66	31	10
BoxI	85	84	30	8
BoxJ	71	32	25	3
BoxK	36	34	25	10
BoxL	97	67	62	7
BoxM	33	25	23	7
BoxN	95	27	26	10
BoxO	94	81	44	9

测试结果 E5-2：

```
utility rate of space is 82.01%
used 0.91 seconds
```

空间利用率为 82.01%，用时 0.91 秒。

测试用例 E5-3：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	49	25	21	13
BoxB	60	51	41	9
BoxC	103	76	64	8
BoxD	95	70	62	6
BoxE	111	49	26	10
BoxF	74	42	40	4
BoxG	85	84	72	10
BoxH	48	36	31	10
BoxI	86	76	38	12
BoxJ	71	48	47	14
BoxK	90	43	33	9
BoxL	98	52	44	9
BoxM	73	37	23	10
BoxN	61	48	39	14
BoxO	75	75	63	11

测试结果 E5-3：

```
utility rate of space is 87.14%
used 1.36 seconds
```

空间利用率为 87.14%，用时 1.36 秒。

测试用例 E5-4：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	97	81	27	6
BoxB	102	78	39	6
BoxC	113	46	36	15
BoxD	66	50	42	8
BoxE	101	30	26	6
BoxF	100	56	35	7
BoxG	91	50	40	12
BoxH	106	61	56	10
BoxI	103	63	58	8
BoxJ	75	57	41	11
BoxK	71	68	64	6
BoxL	85	67	39	14
BoxM	97	63	56	9
BoxN	61	48	30	11
BoxO	80	54	35	9

测试结果 E5-4：

```
utility rate of space is 82.10%
used 1.61 seconds
```

空间利用率为 82.10%，用时 1.61 秒。

测试用例 E5-5：

物品名称	长度	宽度	高度	个数
Bin	587	233	220	1
BoxA	113	92	33	8
BoxB	52	37	28	12
BoxC	57	33	29	5
BoxD	99	37	30	12
BoxE	92	64	33	9
BoxF	119	59	39	12
BoxG	54	52	49	8
BoxH	75	45	35	6
BoxI	79	68	44	12
BoxJ	116	49	47	9

BoxK	83	44	23	11
BoxL	98	96	56	10
BoxM	78	72	57	8
BoxN	98	88	47	9
BoxO	41	33	31	13

测试结果 E5-5：

```
utility rate of space is 78.87%
used 1.43 seconds
```

空间利用率为 78.87%，用时 1.43 秒。

3.4.6 总结

在箱子种类数不同的情况下，方案空间利用率和用时汇总有下表：

箱子种类数	测试编号	空间利用率	用时
三种箱子	E1-1	80.46%	0.77s
	E1-2	74.18%	0.4s
	E1-3	74.18%	0.4s
	E1-4	79.18%	2.76s
	E1-5	77.88%	1.52s
五种箱子	E2-1	82.23%	0.27s
	E2-2	86.64%	0.59s
	E2-3	77.96%	0.84s
	E2-4	80.56%	1.03s
	E2-5	78.88%	0.39s
八种箱子	E3-1	77.82%	0.31s
	E3-2	78.78%	1.47s
	E3-3	83.84%	0.83s
	E3-4	86.83%	0.66s
	E3-5	83.60%	2.78s
十种箱子	E4-1	87.67%	0.79s
	E4-2	78.82%	1.30s
	E4-3	80.65%	1.45s
	E4-4	84.29%	2.84s
	E4-5	82.68%	1.06s
十五种箱子	E5-1	81.21%	0.55s
	E5-2	82.01%	0.91s

	E5-3	87.14%	1.36s
	E5-4	82.10%	1.61s
	E5-5	78.87%	1.43s
平均	-	81.14%	1.13s

方案的平均空间利用率为 81.14%，平均用时为 1.13s。整体上来说，方案的空间利用率较高，同时平均用时较短，是比较高效的方案。同时方案能够满足以下所有约束条件:

（1）被装载的物品必须完全被包含在容器中。

（2）任何两个被装载的物品不能互相重叠。

（3）所有被装载的物品以与容器平行的方式放置，即不能斜放。

（4）静态装箱：从 n 个物品中选取 m 个物品，并且实现 m 个物品在容器内的摆放。

（5）稳定约束：每个被装载的物品必须得到容器底部或者其他物品的支撑，避免被装载的物品悬空。

（6）浮点尺寸：物品的尺寸精度最多考虑到小数点后两位。

（7）顺序摆放：物品在容器内摆放时，按照从内到外或者是从下到上的顺序摆放。

（8）摆放姿态：对于一个 L×W×H 的物品而言，其在容器中有 6 种不同的摆放姿态。

（9）实用性：装箱算法能够较快地实现，在 2 秒以内完成。