国家网络安全学院

课程名称：高级算法

专业年级：2022 级网络空间安全

姓名：梁子游　赵周乔

学号：2022202210134　2022202210128

报告题目：三维装箱算法

1 问题描述

物流公司在流通过程中，需要将打包完毕的箱子装入到一个货车的车厢中，为了提高物流效率，需要将车厢尽量填满，显然，车厢如果能被 100%填满是最优的，但通常认为，车厢能够填满 85%，可认为装箱是比较优化的。

设车厢为长方形，其长宽高分别为 L，W，H；共有 n 个箱子，箱子也为长方形，第 i 个箱子的长宽高为 li, wi, hi (n 个箱子的体积总和是要远远大于车厢的体积)，做以下假设和要求：

1.长方形的车厢共有 8 个角，并设靠近驾驶室并位于下端的一个角的坐标为（0,0,0），车厢共 6 个面，其中长的 4 个面，以及靠近驾驶室的面是封闭的，只有一个面是开着的，用于工人搬运箱子；

2.需要计算出每个箱子在车厢中的坐标，即每个箱子摆放后，其和车厢坐标为（0,0,0）的角相对应的角在车厢中的坐标，并计算车厢的填充率。

2 基本数据结构描述

在装箱问题中，由于所有的物体都是与坐标轴平行的长方体，对它们描述都是通过参考点和各个维度上的边长来指定。参考点就是一个物体的左后下角，也就是其 x、y、z 均最小的点。本文中的装载是在剩余空间中进行的，剩余空间是容器中的未填充长方体空间，其中 x,y,z 描述了它的参考点，lx,ly,lz 描述了它在 3 个维度上的长度。箱子是问题中被装载的物体，我们用 lx,ly,lz 三个域来描述它 3 条边的长度。在这里不同朝向的相同箱子被当作不同箱子处理，箱子有一个域 type 指定了箱子的真实类型。现在我们可以给出问题实例的描述，三元组 (container,box_list,num_list)描述了一个三维装箱问题实例，其中 container 是初始

剩余空间，box_list 是一个箱子向量，指定可用于装载的箱子，num_list 则是一个整数向量，描述了每一种类型箱子的数目。

块和块表

块是基于块装载的启发式算法中装载的最小单位，它是包含许多箱子的长方体。块结构中的每个箱子的摆放都满足约束 C1，而且除了最底部的箱子外都满足 C2。块结构用 lx,ly,lz 描述三条边的长，volume 描述其中箱子的总体积，整数向量 require_list 描述了块对各种类型箱子的需求数。由于块中有空隙，块的顶部有一部分可能由于失去支撑而不能继续放置其它块，我们通过可行放置矩形来描述块的顶部可以继续放置其它块的矩形区域。这里我们仅考虑包括块顶部左后上角的可行放置矩形，以块结构的域 ax,ay 表示其长宽。块表 block_table 是预先生成的按块体积降序排列的所有可能块的列表，用于迅速生成指定剩余空间的可行块列表。同时，块表将块生成算法与装载算法分开，使得更换块生成算法变得容易。

剩余空间堆栈和剩余箱子

在基础启发式算法中，剩余空间被组织成堆栈。算法基本过程可描述为：从栈顶取一个剩余空间，若有可行块，按照装载序列选择一个块放置在该空间，将未填充空间切割成新的剩余空间加入堆栈，若无可行块则抛弃此剩余空间，如此反复直至堆栈为空。在此过程中，space_stack 表示剩余空间堆栈，整数向量 avail_list 记录各种剩余箱子的数目。

放置

一个放置是一个剩余空间和块的二元组。例如(space,block) 表示将块 block 放置在剩余空间 space 上。放置是通过将块的参考点和剩余空间的参考点重合得

到的。由于算法要保证剩余空间总是被支撑的，而块中除了底部箱子，所有的箱子都满足稳定性约束，也就是顶部面积大于待放置的箱子面积。

3 块生成算法

简单块生成：

简单块是由同一朝向的同种类型的箱子堆叠而成的，箱子和箱子之间没有空隙，堆叠的结果必须正好形成一个长方体。下图展示了两个简单块的例子，其中 nx,ny,nz 表示在每个维度上的箱子数。$nx \times ny \times nz$ 则是简单块所需要的总箱子数执行。如下图所示：

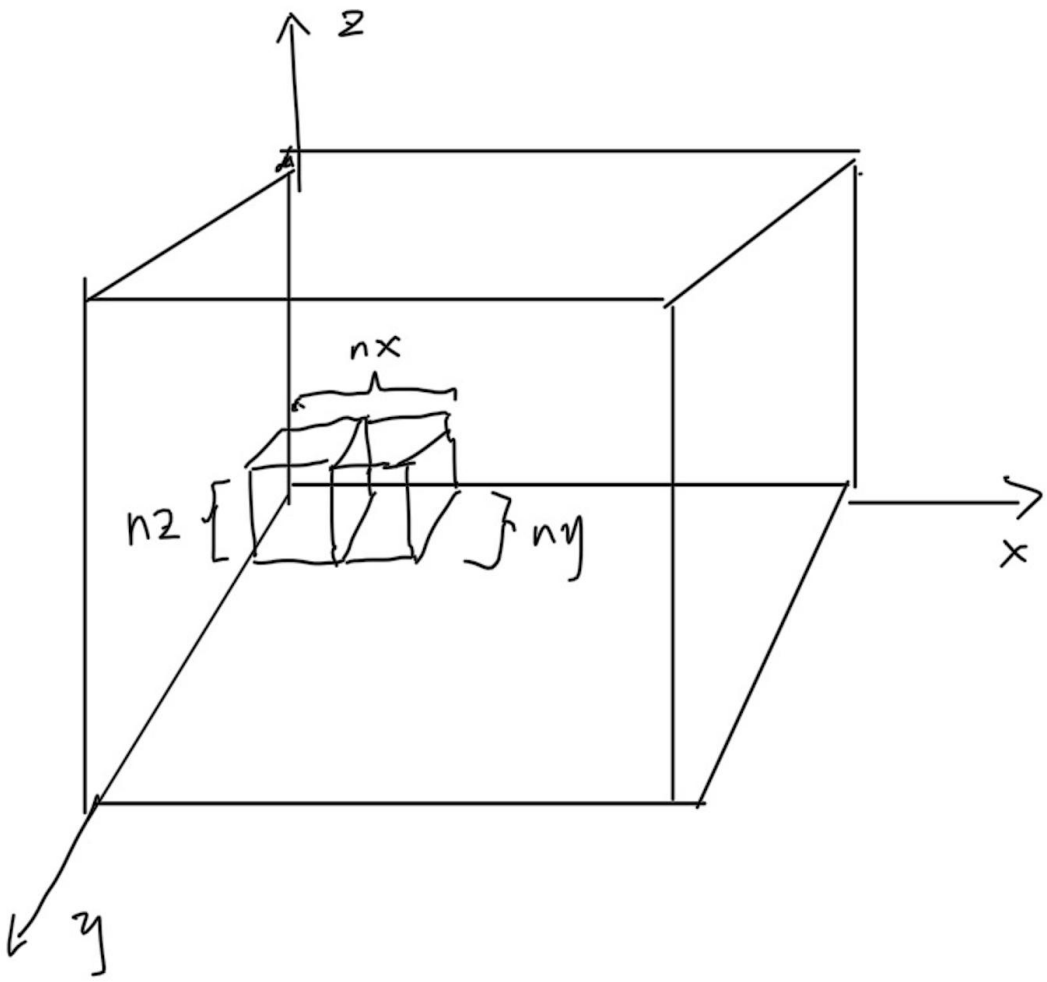


图 1 简单块生成

复合块生成

复合块是通过不断复合简单块而得到的，我们定义复合块如下：

(1) 简单块是最基本的复合块。

(2) 有两个复合块 a 和 b。可以按 3 种方式进行复合得到复合块 c ：按 x 轴方向复合，按 y 轴方向复合，按 z 轴方向复合。c 的大小是包含 a 和 b 的最小长方体。当然复合块会很多，因此我们需要设置一个最大复合块个数。

复合块的基本思想是组成类似于积木一样的块，便于后续的算法镶嵌。但是复合块也遵循一定的原则。具体如下表所示：

方向	a,b 需要满足的条件	复合块 c 的属性	复合块 c 满足的条件
x 轴	a.ax=a.lx b.ax=b.lx a.lz=b.lz	c.ax=a.ax+b.ax c.ay=min(a.ay,b.ay) c.lx=a.lx+b.lx c.ly=max(a.ly,b.ly) c.lz=a.lz	c.lx<+container.x c.ly<=container.y c.lz<=container.lz c.require<=num c.volume/(c.lx*c.ly*c.lz) >=MinFillRate c.ax*c.ay/(c.lx*c.ly) >=MinAreaRate c.times<=MaxTimes
y 轴	a.ay=a.ly b.ay=b.ly a.lz=b.lz	c.ax=min(a.ax,b.ax) c.ay=a.ay+b.ay c.lx=max(a.lx,b.lx) c.ly=a.ly+b.ly c.lz=a.lz	
z 轴	a.ax>=b.lx a.ay>=b.ly	c.ax=b.ax c.ay=b.ay c.lx=a.lx c.ly=a.ly c.lz=a.lz+b.lz	
通用		c.require=a.require+b.require c.volume=a.volume+b.volume c.childs:(a,b) c.times=max(a.times,b.times)+1	

制定这样的原则在于，我们想复合块其上面也能放东西，也就是也能满足支撑性需求。如下图所示，这样复合后的块，上面依然可以连着放一堆东西。不然随意进行复合后的块，其上面由于难以放置其他物品，就会导致效率很低下。还不如使用简单块。

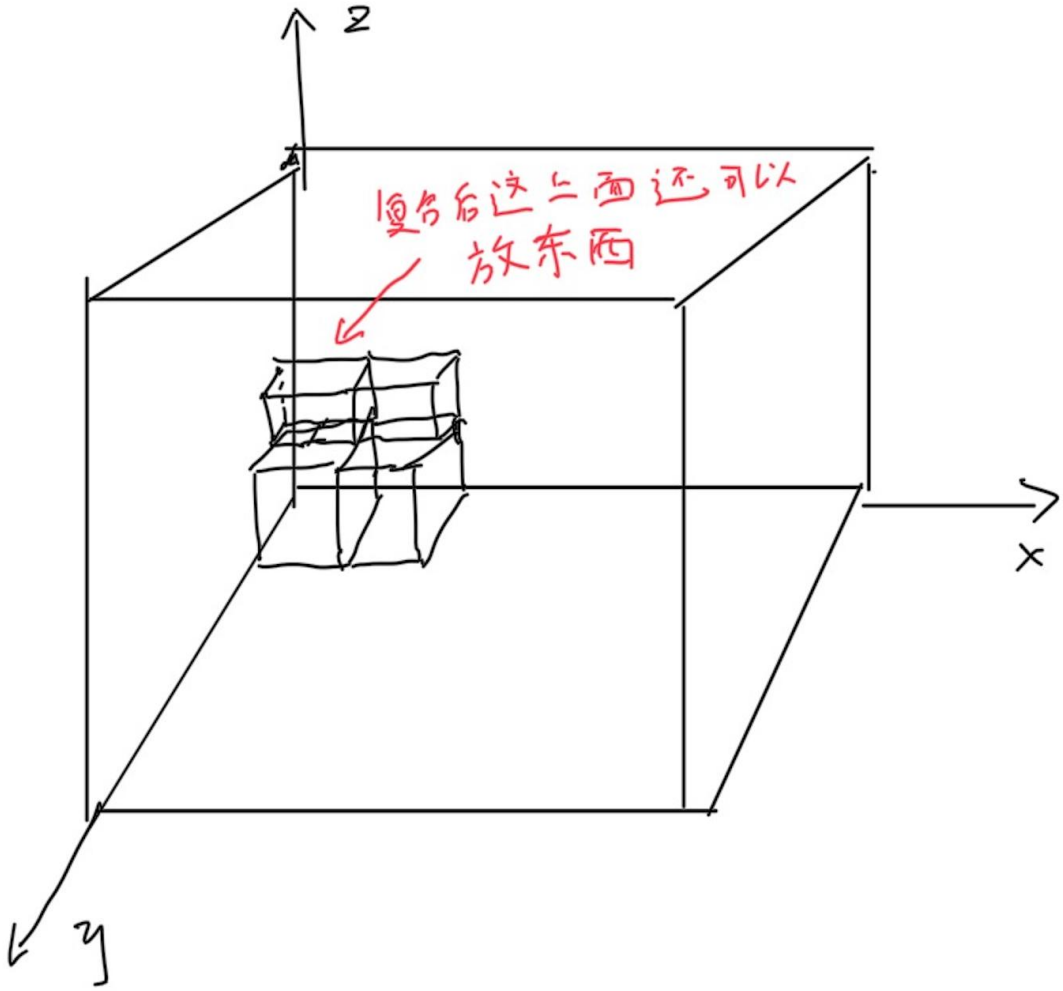


图 2 复合块生成

4. 可行块生成

可行块生成算法 gen_block_list(space,avail,block_table)用于从 block_table 中获取适合当前剩余空间的可行块列表。该算法扫描 block_table，返回所有能放入剩余空间 space 并且 avail 有足够剩余箱子满足 require 的块。由于 block_table 是按

块中箱子总体积降序排列的，返回的 block_list 也是按箱子总体积降序排列的，后续也会优先选择体积大的，有点类似于贪心算法。

5 空间切割和空间转移

在每个装载阶段一个剩余空间被装载，装载分为两种情况：有可行块，无可行块。在有可行块时，算法按照块选择算法选择可行块，然后将未填充空间切割成新的剩余空间。在无可行块时，当前剩余空间被抛弃，若其中的一部分空间可以被并入当前堆栈中的其他空间，则进行空间转移重新利用这些空间。下图显示了在考虑稳定性约束剩余空间与块结合成为放置以后的状态。未填充空间将被按照不同情况，沿着块的三个面被分成三个剩余空间。需要注意的是，在考虑稳定性约束时，由于要保证所有剩余空间受到足够的支撑，z zz 轴上的新剩余空间在切割的时候必须沿着所选块的顶部可放置矩形进行。因此，块顶部的可放置矩形确定了一个新的剩余空间 paceZ ，其余的两个剩余空间为 x 轴方向上的 spaceX 和 y 轴方向上的 spaceY ，所以只有两种切割方法。如下面两张图所示：

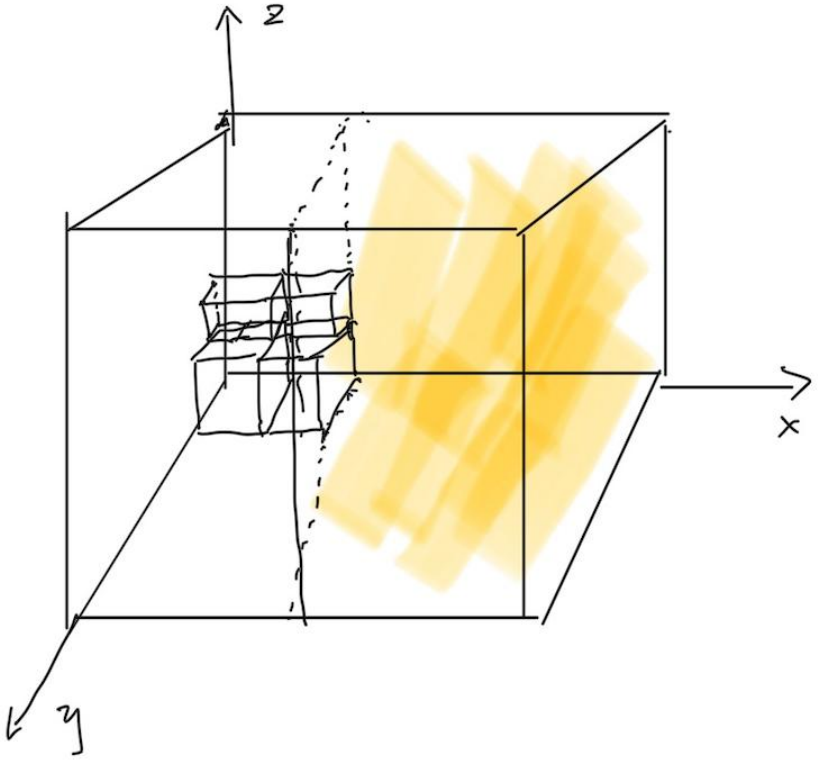


图 3 切割方法 1

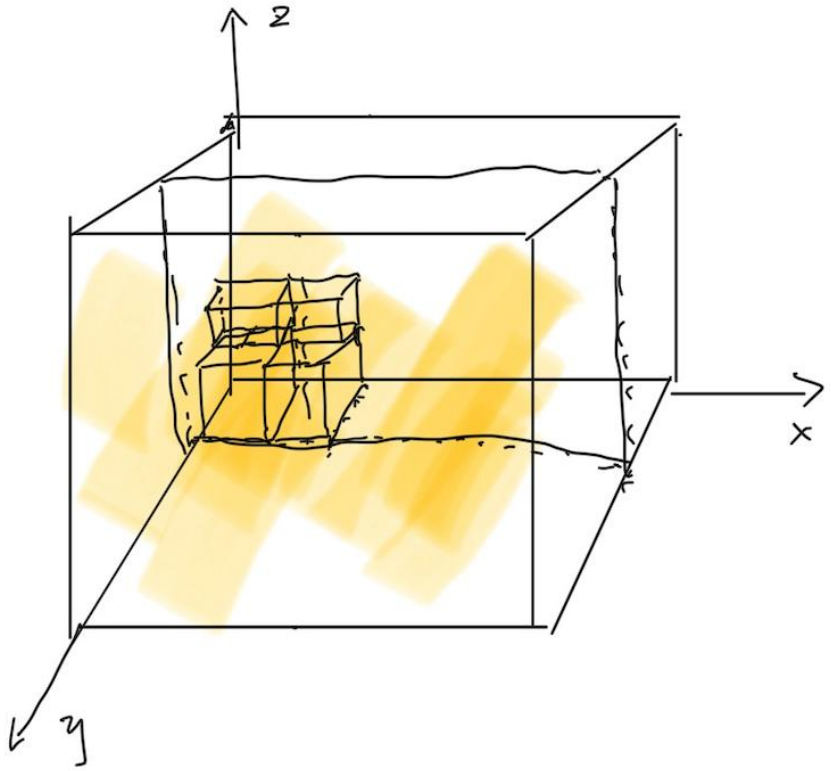


图 4 切割方法 2

各种切割方式本质上的不同就在于可转移空间的归属。我们希望在切割过程中尽量保证空间完整性，而衡量空间完整性的方法有很多，我们选择的策略是另切割出的剩余空间尽可能的大。这里大小的判定以剩余空间在放置块以后在 x 轴、y 轴和 z 轴上的剩余长度 mx、my、mz 作为度量，将**可转移空间**分给剩余量较大的方向上的新空间。可转移空间就是经过基本切割之后，几部分空间交叉的部位如下图所示。算法中 gen_residual_space(space,block)执行未填充空间的切割，其返回的剩余空间 spaceX、spaceY、spaceZ 按照 mx、my、mz 的从小到大排列，并确保最后入栈的是包含可转移空间的剩余空间。

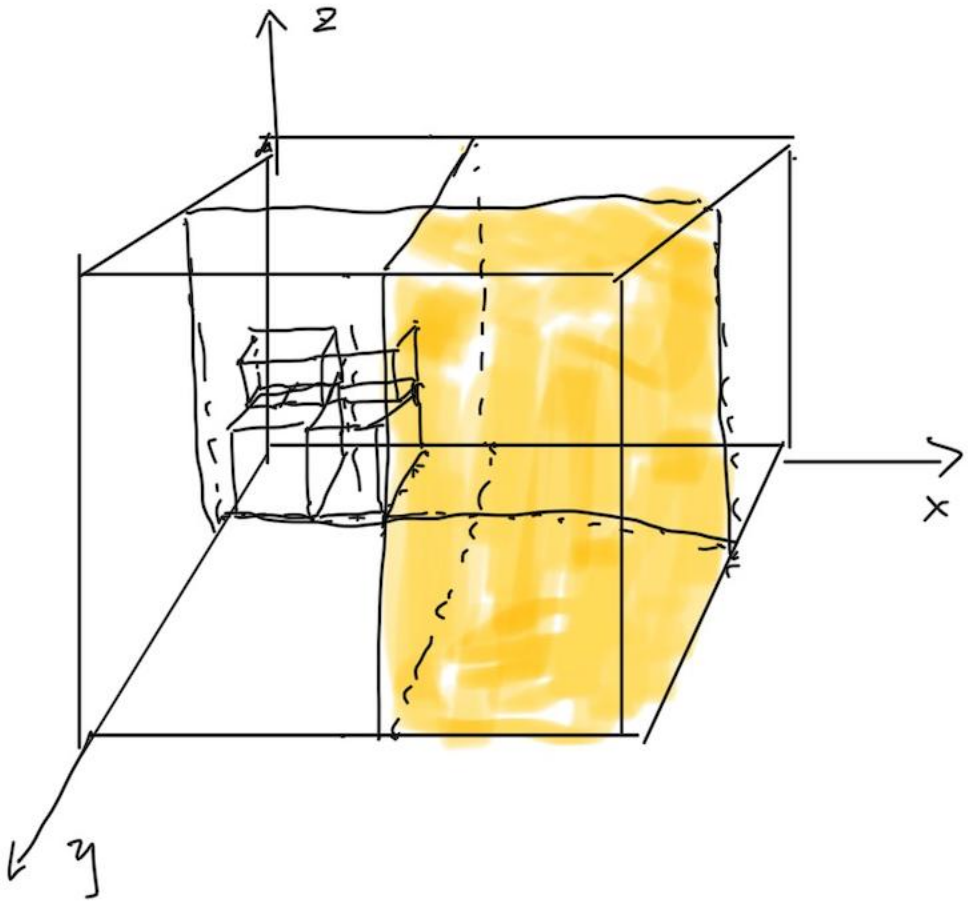


图 5 可转移空间示意

由于切割算法保证了包含可转移空间的剩余空间后入栈，所以其必然被先装载，若在装载过程中可行块列表为空，栈顶空间中的可转移空间可以被转移给剩

余空间堆栈中来自同一次切割的其他空间以重新利用。因此，我们可以通过重新切割未填充空间来达到再次利用可转移空间的目的，算法中的 transfer_space(space,space_stack) 就是执行这样的任务，此过程判定当前剩余空间与栈顶的一个或两个剩余空间是否是由同一次切割而产生的，若是则将可转移空间转给相应的一个或两个剩余空间。

6 块选择算法

遍历整个可行块列表，尝试放置当前块到当前部分放置方案，然后用某种方式评估此状态，并将此评估值作为被选块的适应度，最终选取适应度最高的块作为结果。

整体流程如下图所示。

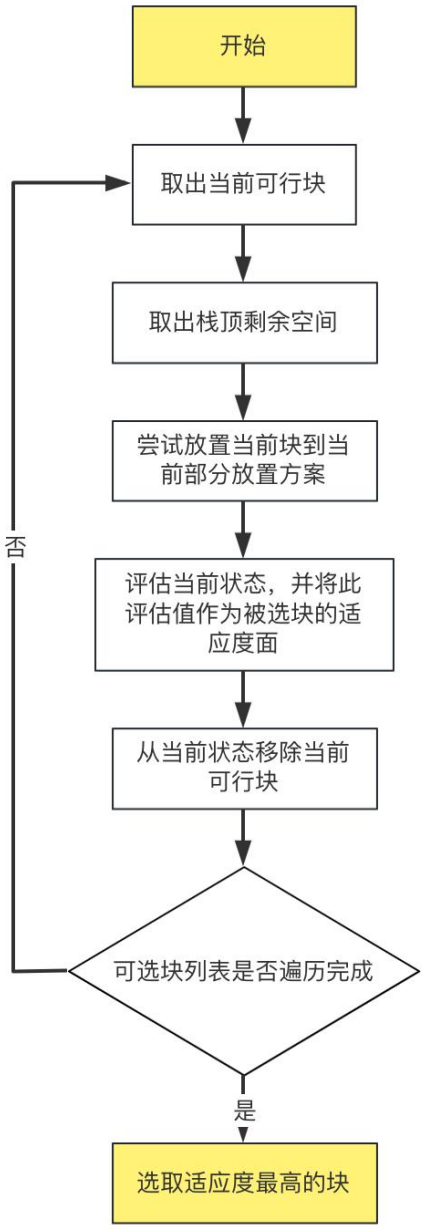


图 6 块选择算法整体流程

块放置和块移除算法

块放置算法完成的工作包括将块和栈顶空间结合成一个放置加入当前放置方案，移除栈顶空间，扣除已使用物品，然后切割未填充空间并加入剩余空间堆栈。

块移除算法完成的工作包括从当前部分放置方案中移除当前块所属的放置，恢复已使用物品，移除空间堆栈栈顶的三个切割出来的剩余空间，并将已使用剩余空间重新插入栈顶。

补全算法

除了块放置和块移除算法，另一个极其重要的算法是部分放置方案补全算法。我们知道，评估当前的部分放置方案好坏的最直接的方法是用某种方式补全它，并以最终结果的填充率作为当前状态的评估值。该算法实际上是整体基本启发式算法的一个简化版本，区别在于每个装载阶段算法都选择可行块列表中体积最大的块进行放置。由于可行块列表已经按照体积降序排列，实际上算法选择的块总是列表的第一个元素。算法不改变输入的部分放置方案，只是把最终补全的结果记录在此状态的 volume_complete 域作为该状态的评估值。

深度优先搜索算法

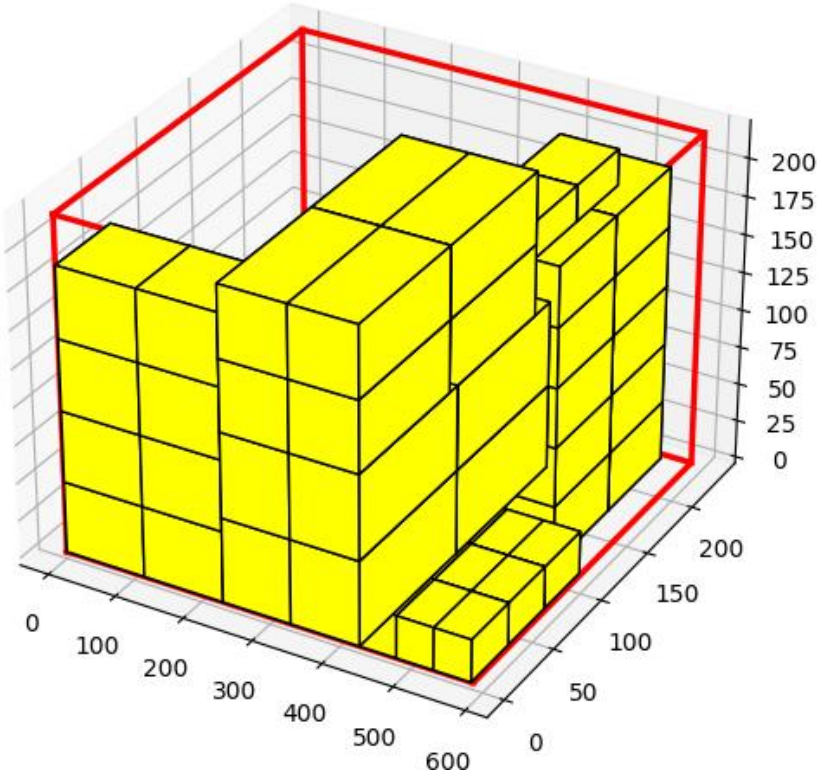
我们采用了深度优先算法来进行遍历选择，然而我们知道，由于块太多，无限深度优先遍历会导致路径爆炸，从而无法得到结果，因此我们需要限制深度。在到达 max 深度时进行补全算法估算，最终综合选择效果最佳的选块方案。

7 部分测试

经过一些参数调整，我们得到了比较好的结果。

对于 15 种箱子的 E5-5,能达到 43.16%的准确率

```
剩余未装入的箱子: [8, 3, 5, 12, 9, 2, 8, 6, 12, 1, 9, 2, 8, 1, 13]
已装入车厢的体积: 12985622.0
总箱体填满的比例: 43.16%
2022-12-31 14:07:03.758 Python[16495:5542196] +[CATransaction synchro
```



对于测试数据 E1-1，我们能达到百分之 75 的填满率。这组数据的参数调整核心在于只使用简单块，不考虑复合块。

```python
# E1-1
container = Space(0, 0, 0, 587, 233, 220)
box_list = [Box(108, 76, 30, 0), Box(110, 43, 25, 1), Box(92, 81, 55, 2)]
num_list = [40, 33, 39]
test(container, box_list, num_list,0)
```

Python

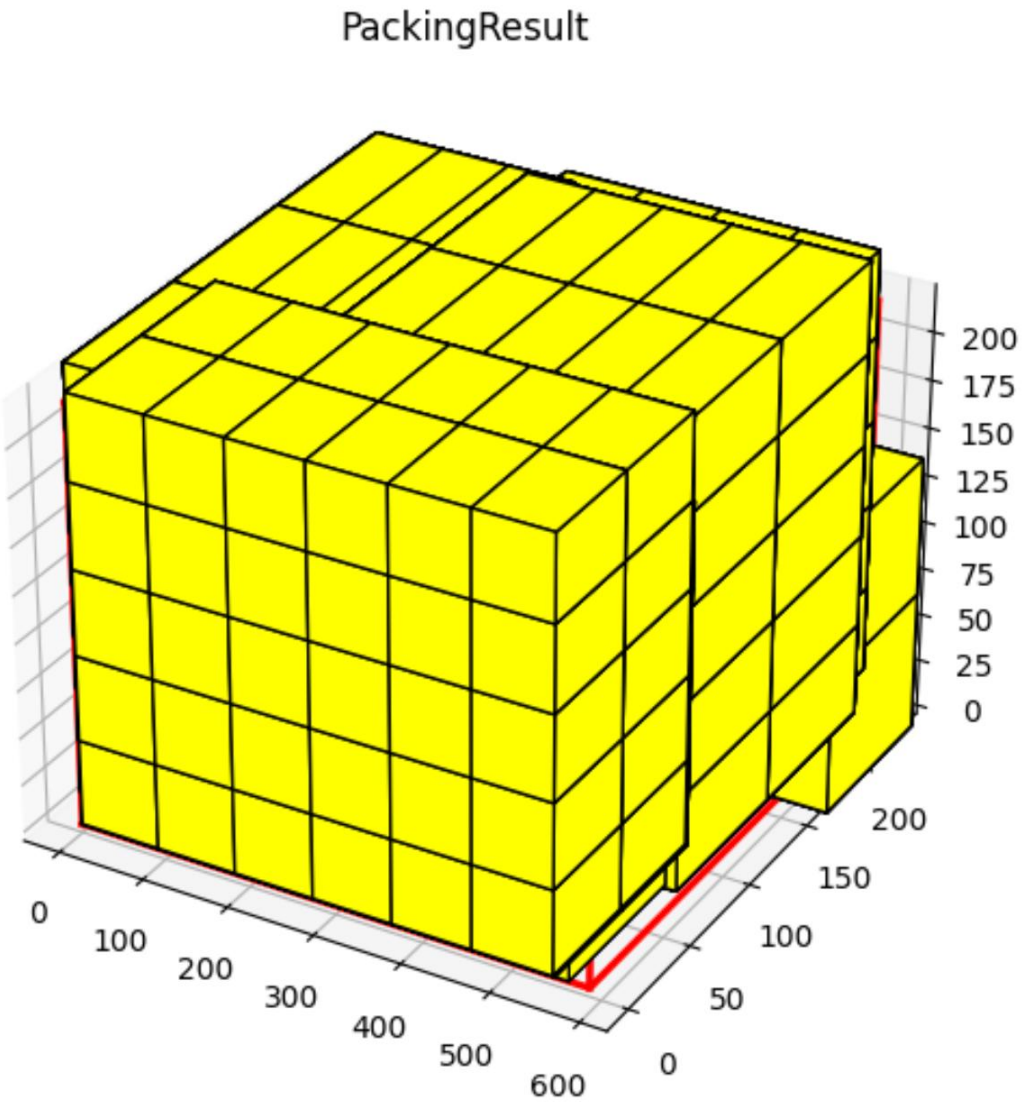
```
剩余未装入的箱子: [1, 33, 7]
已装入车厢的体积: 22718880.0
总箱体填满的比例: 75.5%
```

对于测试数据 E1-2 能达到 68%的填满率。

```python
# E1-2
container = Space(0, 0, 0, 587, 233, 220)
box_list = [Box(91, 54, 45, 0), Box(105, 77, 72, 1), Box(79, 78, 48, 2)]
num_list = [32, 24, 30]
test(container, box_list, num_list,1)
```

Python

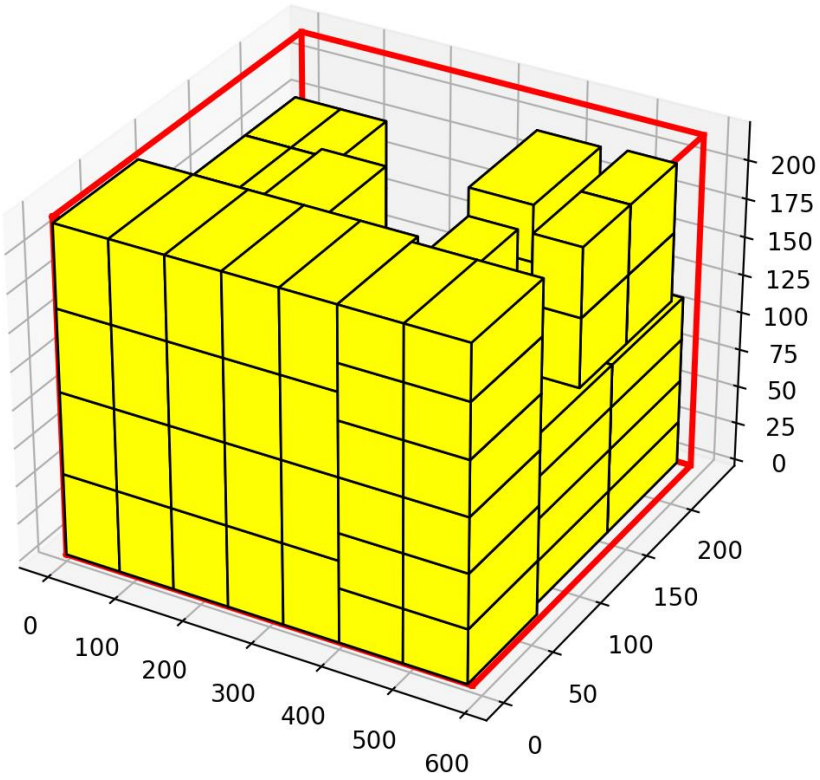
```
剩余未装入的箱子: [0, 6, 20]
已装入车厢的体积: 20512080.0
总箱体填满的比例: 68.17%
```



对于 5 种箱子的 E2-4 能达到百分之 70.86%的准确率：

```
# E2-4
container = Space(0, 0, 0, 587, 233, 220)
box_list = [Box(90, 70, 36, 0), Box(84, 78, 28, 1), Box(94, 85, 39, 2), Box(80, 76, 54, 3), Box(69, 50, 45, 4)]
num_list = [16, 28, 20, 23, 31]
test(container, box_list, num_list,1)
```

```
剩余未装入的箱子: [1, 0, 15, 3, 1]
已装入车厢的体积: 21320718.0
总箱体填满的比例: 70.86%
```



对于 5 种箱子的 E2-5 能达到 73.23%的准确率：

```
# E2-5
container = Space(0, 0, 0, 587, 233, 220)
box_list = [Box(74, 63, 61, 0), Box(71, 60, 25, 1), Box(106, 80, 59, 2), Box(109, 76, 42, 3), Box(118, 56, 22, 4)]
num_list = [22, 12, 25, 24, 11]
test(container, box_list, num_list,1)
```

Python

```
剩余未装入的箱子: [0, 0, 13, 0, 10]
已装入车厢的体积: 22033892.0
总箱体填满的比例: 73.23%
```

8 总结

本学期高级算法课程受益匪浅，先是复习了一些基本的算法知识，对之前学过的知识进行的复习，接着学习了线性规划相关算法，学习了如何求解线性方程组中的相等与不等的问题，了解了单纯形法等算法。接着学习了高级图算法，对最大流最小割问题、图的中心性算法问题、社群发现问题进行了探讨，了解了目前比较前沿的一些算法思想。接着学习了 NP 问题的相关证明方法，这部分主要是学习如何归约，也就是当一个问题我们不知道其难度时，我们可以进行转化，

运用转化思想，归约成一个已经存在的 npc 问题。最后还学习了近似算法、随机算法、启发式算法等，当我们很难获得一个确定性解时，我们可以考虑使用近似算法和随机算法等，或许能获得比较好的解决方案。

在本次大作业中，我们结合了深度遍历、启发式算法、分割子问题递归等思想。将问题一点一点进行分解，最后结合起来的过程是比较漫长的，中间也出现了很多问题（包括参数设置不当，算法效果不好等）。最终也获得了一个较为满意的效果，对大部分 case 都能达到百分之 50 以上的填充率，部分箱子比较“适配”的，还能达到百分之 75 以上的填充率。