

武汉大学国家网络安全学院

实验报告

课程名称 高级算法设计与分析

专业年级 网络空间安全 2022

姓 名 宋啟鹏

学 号 2022202210012

协 作 者 无

实验学期 2022-2023 学年 第一 学期

填写时间 2022 年 12 月 30 日

实验介绍

【实验名称】： 物流装箱问题基础

【实验介绍】：

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 l_i , w_i , h_i (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。

基础部分要求：

1. 所有的参数为整数；
2. 静态装箱，即从 n 个箱子中选取 m 个箱子，并实现 m 个箱子在车厢中的摆放（无需考虑装箱的顺序，即不需要考虑箱子从内向外，从下向上这种在车厢中的装箱顺序）；
3. 所有的箱子全部平放，即箱子的最大面朝下摆放；
4. 算法时间不做严格要求，只要 1 天内得出结果都可。

【实验环境】：

(1) 硬件资源： 一台 **Windows** 个人电脑

(2) 软件资源： **PyCharm**

实验内容

【实验方案设计】：

- 1、创建箱子类，用来表示每个箱子的数据，箱子类有成员 x 、 y 、 z 分别箱子的长宽高。
- 2、箱子应该满足个约束：1) 一条边表示箱子的方向，如果去除这个约束则表示三个边都能作为高度。2) 稳定约束。箱子不能悬空，出了最底部的箱子外所有箱子下方都应该有箱子。
- 3、块是多个箱子的复合体，也用长宽高来描述，同时还有体积和对各个种类箱子的数量。当块中存在间隙导致部分箱子没有支撑的时候描述顶部的区域来查看是否能放其他箱子。
- 4、剩余空间与箱子。将剩余的空间组织为堆栈。当要放入箱子的时候从堆栈顶部取出一个剩余空间，然后将箱子放入该空间中，将该空间的未填充的部分切割为新的剩余空间并加入堆栈。如果没有找到能放入该空间的块就抛弃这一个剩余空间，重复直到堆栈为空。
- 5、放置方案是一个剩余空间和块的二元组， (x, y) 表示将块 y 放置在剩余空间 x 上。通过将块的参考点和剩余空间的参考点重合得到一个放置。在算法的某一个时刻，剩余空间堆栈和剩余物品构成了一个部分放置方案。
- 6、生成块。块分为简单块和复合块。简单块是同一类型的箱子堆叠而成，之间没有空隙，正好能堆成一个长方体。复合块为简单块的组合。两种类型的简单块有三种组合方法，即小块分别在大块的 x 、 y 、 z 三个方向堆叠。由于复合块的数量非常多，需要对其施加限制：1) 大小不能大于容器 2) 可以有空隙，但必须达到一定的填充率 3) 复合之后的块也需要满足简单块的条件 4) 复合块的复杂度

需要进行标记, 记简单块为 0, 复合块为子块复杂度加一。5) 三边长度相同、箱子需求相同的复合块被视作相同的块, 重复生成的相同块被忽略。

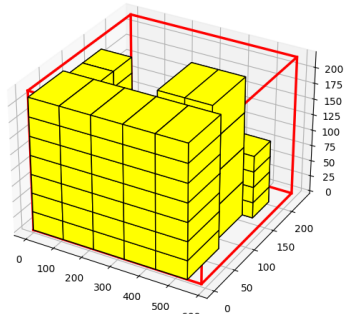
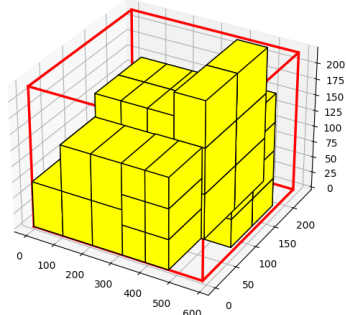
7、空间切割和转移。在每个装载阶段一个剩余空间被装载, 装载分为两种情况: 有可行块, 无可行块。在有可行块时, 算法按照块选择算法选择可行块, 然后将未填充空间切割成新的剩余空间。在无可行块时, 当前剩余空间被抛弃, 若其中的一部分空间可以被并入当前堆栈中的其他空间, 则进行空间转移重新利用这些空间。

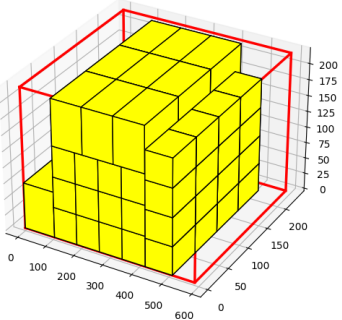
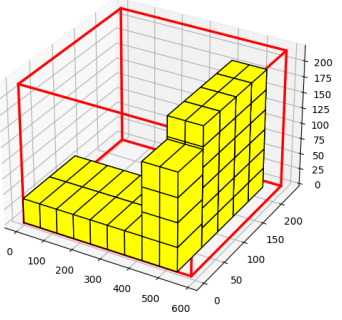
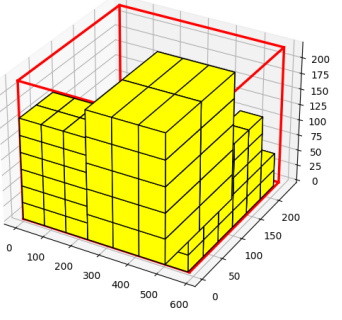
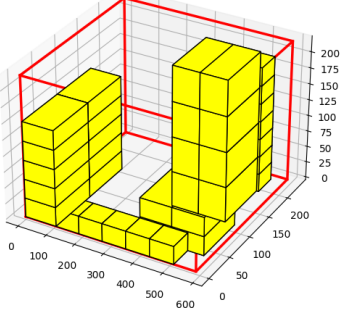
8、块的选择算法。1) 整体流程为遍历整个可行块列表, 尝试放置当前块到当前部分放置方案, 然后用某种方式评估此状态, 并将此评估值作为被选块的适应度, 最终选取适应度最高的块作为结果。2) 块放置算法完成的工作包括将块和栈顶空间结合成一个放置加入当前放置方案, 移除栈顶空间, 扣除已使用物品, 然后切割未填充空间并加入剩余空间堆栈。块移除算法完成的工作包括从当前部分放置方案中移除当前块所属的放置, 恢复已使用物品, 移除空间堆栈栈顶的三个切割出来的剩余空间, 并将已使用剩余空间重新插入栈顶。3) 除了块放置和块移除算法, 另一个极其重要的算法是部分放置方案补全算法。我们知道, 评估当前的部分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是否装填完成域作为该状态的评估值。4) 贪心策略。在块的选择算法过程中, 采用贪心算法, 直接返回填充体积最大的块, 由于可行块列表已经按照体积降序排

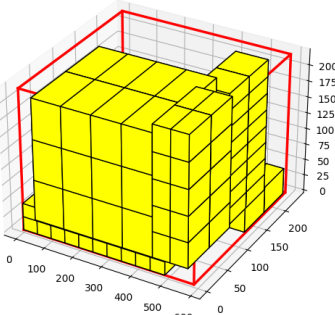
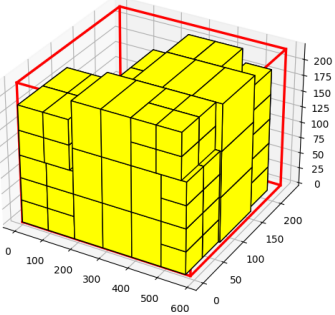
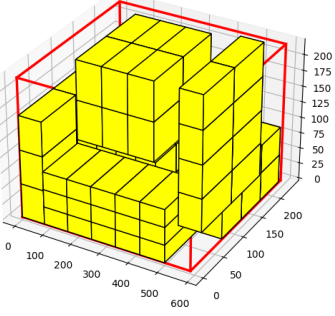
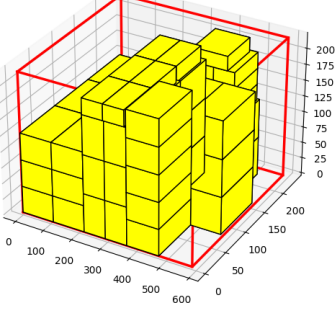
列，实际上算法选择的块总是列表的第一个元素。5) 深度优先搜索策略。由于每个阶段可行块列表包含大量的块，算法往往也限制每个节点的最大分支数。采用深度优先搜索算法扩展当前放置方案，算法的输入为一个部分放置方案，深度限制和最大分支数。该算法从一个部分放置方案出发，递归的尝试可行块列表中的块，在到达深度限制的时候调用补全函数得到当前方案的评估值，并记录整个搜索过程找到的最优的评估值作为输入部分放置方案的评估。

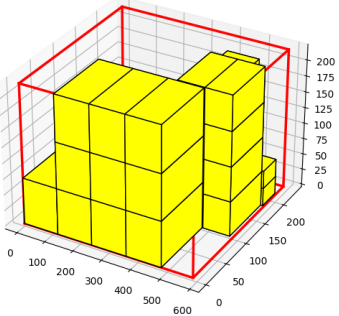
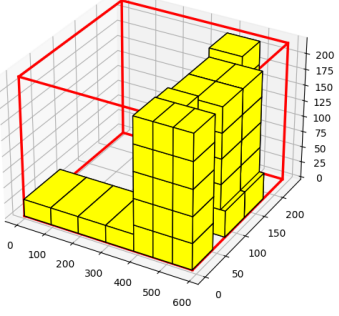
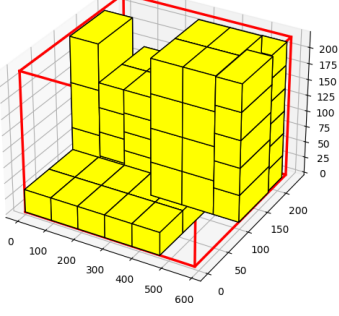
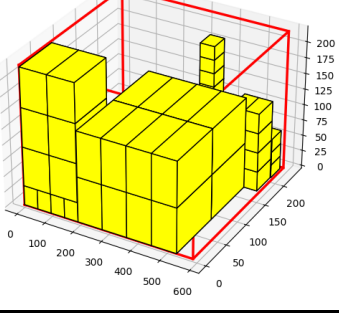
【实验结果分析】：

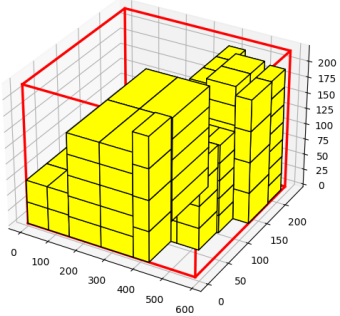
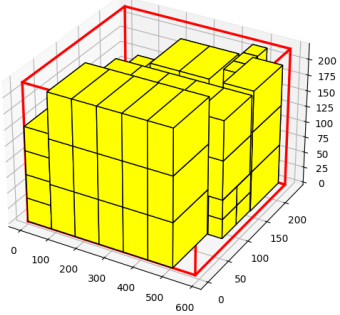
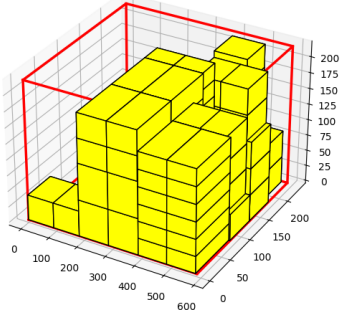
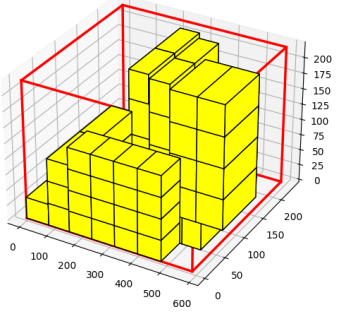
使用程序运行测试数据的结果如下图表所示：

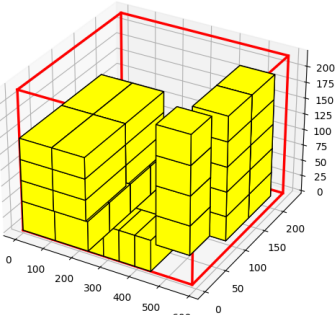
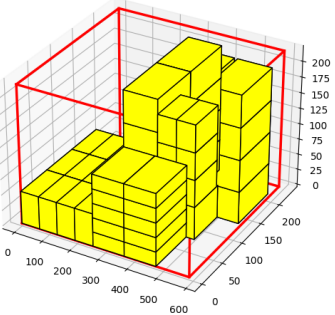
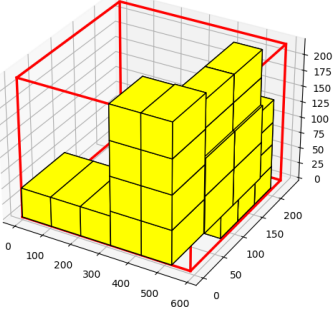
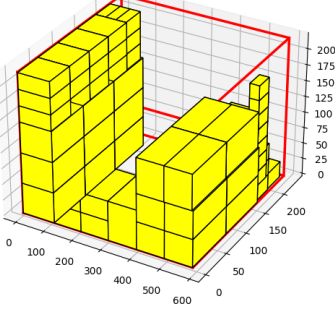
例子	空间占用	空间占比	花费时间(秒)	放置结果
E1-1	16538250	54.9633%	27.8075	
E1-2	16418376	54.5649%	11.4565	

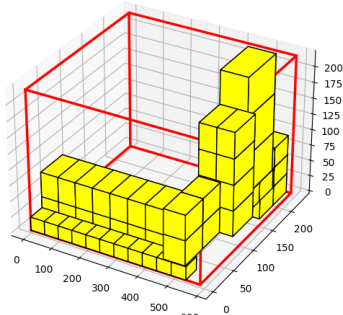
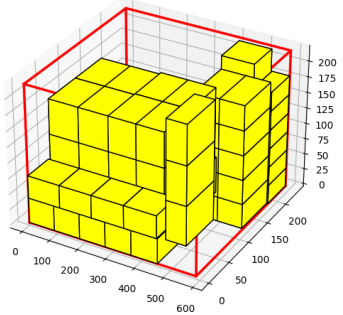
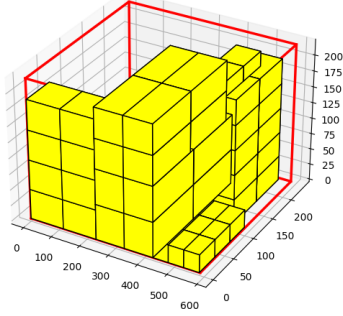
E1-3	21114864	70.1732%	11.8965	 <p>A 3D bar chart showing a rectangular block of yellow cubes. The x-axis ranges from 0 to 600, the y-axis from 0 to 200, and the z-axis from 0 to 200. The block is approximately 500 units wide, 150 units deep, and 150 units high.</p>
E1-4	6723720	22.3456%	75.8084	 <p>A 3D bar chart showing a stepped structure of yellow cubes. The x-axis ranges from 0 to 600, the y-axis from 0 to 200, and the z-axis from 0 to 200. The structure has a base of approximately 500x150 and a height of approximately 150.</p>
E1-5	14207886	47.2186%	95.8797	 <p>A 3D bar chart showing a rectangular block of yellow cubes. The x-axis ranges from 0 to 600, the y-axis from 0 to 200, and the z-axis from 0 to 200. The block is approximately 500 units wide, 150 units deep, and 150 units high.</p>
E2-1	8452118	28.0898%	106.3724	 <p>A 3D bar chart showing a stepped structure of yellow cubes. The x-axis ranges from 0 to 600, the y-axis from 0 to 200, and the z-axis from 0 to 200. The structure has a base of approximately 500x150 and a height of approximately 150.</p>

E2-2	17612502	58.5335%	497.6046	
E2-3	20352528	67.6397%	189.4029	
E2-4	19878750	66.0651%	86.512	
E2-5	18684044	62.0946%	30.7933	

E3-1	15275524	50.7668%	173.5374	
E3-2	8424164	27.9969%	443.4249	
E3-3	14263869	47.4046%	2581.057	
E3-4	12971448	43.1094%	1358.673 9	

E3-5	14193769	47.1716%	993.8506	
E4-1	21745921	72.2705%	2712.311 8	
E4-2	15264132	50.7289%	1871.945 0	
E4-3	12610280	41.9091%	321.5568	

E4-4	11137758	37.0153%	1331.808 7	
E4-5	12380470	41.1453%	618.6724	
E5-1	10006189	33.2546%	495.268	
E5-2	11630771	38.6538%	10972.18 44	

E5-3	5155203	17.1328%	42244.19 29	
E5-4	15101738	50.1892%	1677.336 2	
E5-5	12985622	43.1565%	2039.988 5	

【实验总结】：

有几类例子出现了空间利用率极低的情况，这种是由于程序只运行了一次，因为一些随机因素的影响所以造成了程序运行的不稳定，通过将一个例子运行多次后取平均值可以改善这种情况；还可以观察到空间利用率在箱子种类较低的时候容易达到较高，在箱子种类增多的时候更容易出现空间利用率降低较多的情况；同时花费的时间在箱子种类较少的时候较短，在箱子种类增多的时候话费的时间呈逐渐增多的趋势。