实验报告

团队成员 1： 彭君茹 2022202210148

团队成员 2： 何杭薇 2022202210135

2022 年 12 月 15 日

目录

1 实验完成内容 1
2 实验工具 1
3 实验算法原理 1
3.1 jsonfile_generation(excel_path,out_path) 1
3.2 boxes_generation(case) 2
3.3 Stacks_generation(boxes,carriageHeight) 3
3.4 Box_loader(carriage_Size, boxes) 3
3.4.1 基于贪心的装车算法 5
3.4.2 随机算法 6
4 实验结果和实验结果分析 6
4.1 单个实验结果分析 6
4.2 多个实验结果分析 7
5 实验总结 8

1 实验完成内容

基础部分：

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。

高级部分：

1) 参数考虑小数点后两位。
2) 实现在线算法，也就是箱子是按照随机顺序到达，先到达先摆放。

算法需要实时得出结果，即算法时间小于等于 2 秒。

2 实验工具

1) Python 语言。Python 是一种动态类型、常用于人工智能和数据分析的编程语言。
2) Pandas 工具包。Pandas 是一个基于 NumPy 的数据分析包，提供了大量能使我们快速便捷地处理数据的函数和方法。
3) Random 包。Random 是内建（built-in）函数，本实验用 Random.shuffle 将箱子的顺序在序列中进行打乱
4) Json 标准库。JSON 是一种轻量级的数据交换格式。本实验用其进行数据的存储和表示。

3 实验算法原理

3.1 jsonfile_generation(excel_path,out_path)

提供的数据集包含了三种箱子、五种箱子、八种箱子、十种箱子以及十五种箱子的情况，并且每个情况中包含多个测试用例。该函数的主要作用是将不同箱

子种类的.xlsx 格式的测试用例转化成字典形式后存储为 json 格式，方便后续的读取操作。以下以五种箱子为例，存储形式如下：

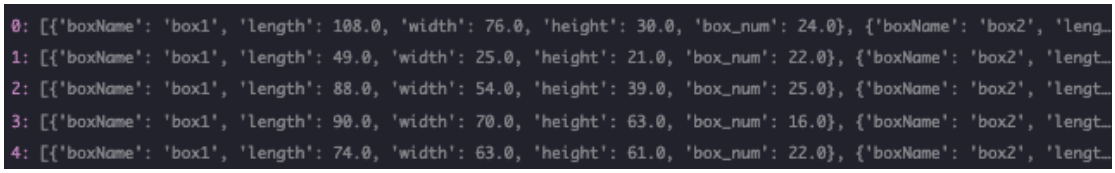

图 3-1 测试用例存储格式

其中每一项所对应一个测试用例，每一个测试用例中包含五种箱子的数据，每个数据有箱子的长宽高以及箱子数量信息。

后续会针对每一个测试用例进行放入箱子的算法。

3.2 boxes_generation(case)

该函数的作用主要是对 3.1 中的 json 文件中的每一个测试用例的箱子数据进行处理，使每一个箱子对应返回值 boxes 中的一项，每一项包括箱子的名字以及尺寸信息。针对五种箱子中的第一个测试用例，返回的 boxes 如下：

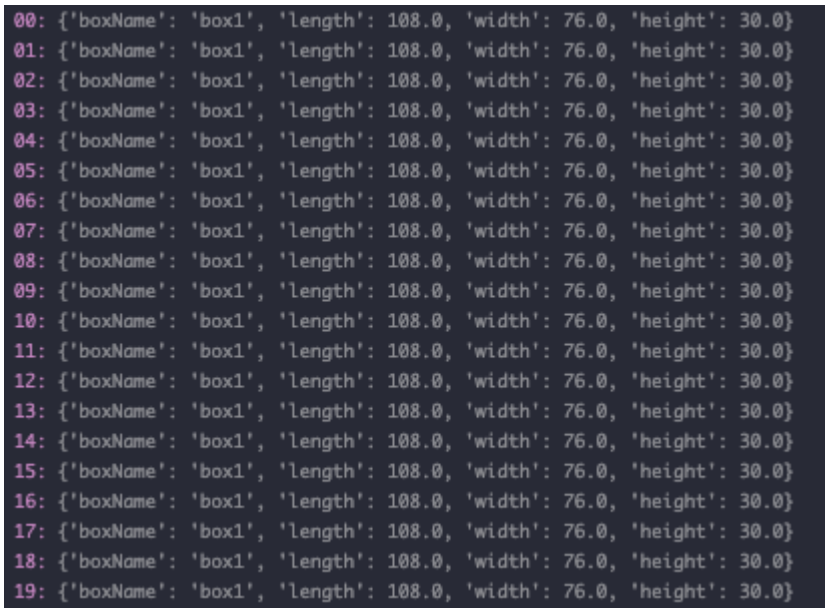

图 3-2 boxes 存储格式

该测试用例中有五种箱子共八十一个，所以返回值中共有八十一项。

3.3 Stacks_generation(boxes,carriageHeight)

该函数的作用是将长与宽相同的箱子进行堆叠成为一个堆垛，堆垛的高度不超过车厢的高度 carriageHeight，所以该函数会返回 n 个高度不超过车厢高度的堆垛，每个堆垛中的箱子的长与宽都是相同的。

同样针对五种箱子中的第一个测试用例，生成的堆垛如下：

```
00: [{'boxName': 'box5', 'length': 120.0, 'width': 99.0, 'height': 73.0}, {'boxName': 'box5
01: [{'boxName': 'box4', 'length': 81.0, 'width': 33.0, 'height': 28.0}, {'boxName': 'box4'
02: [{'boxName': 'box2', 'length': 110.0, 'width': 43.0, 'height': 25.0}, {'boxName': 'box2
03: [{'boxName': 'box1', 'length': 108.0, 'width': 76.0, 'height': 30.0}, {'boxName': 'box1
04: [{'boxName': 'box3', 'length': 92.0, 'width': 81.0, 'height': 55.0}, {'boxName': 'box3'
05: [{'boxName': 'box5', 'length': 120.0, 'width': 99.0, 'height': 73.0}, {'boxName': 'box5
06: [{'boxName': 'box3', 'length': 92.0, 'width': 81.0, 'height': 55.0}, {'boxName': 'box3'
07: [{'boxName': 'box1', 'length': 108.0, 'width': 76.0, 'height': 30.0}, {'boxName': 'box1
08: [{'boxName': 'box3', 'length': 92.0, 'width': 81.0, 'height': 55.0}, {'boxName': 'box3'
09: [{'boxName': 'box5', 'length': 120.0, 'width': 99.0, 'height': 73.0}, {'boxName': 'box5
10: [{'boxName': 'box3', 'length': 92.0, 'width': 81.0, 'height': 55.0}, {'boxName': 'box3'
11: [{'boxName': 'box4', 'length': 81.0, 'width': 33.0, 'height': 28.0}, {'boxName': 'box4'
12: [{'boxName': 'box5', 'length': 120.0, 'width': 99.0, 'height': 73.0}, {'boxName': 'box5
13: [{'boxName': 'box3', 'length': 92.0, 'width': 81.0, 'height': 55.0}, {'boxName': 'box3'
14: [{'boxName': 'box1', 'length': 108.0, 'width': 76.0, 'height': 30.0}, {'boxName': 'box1
15: [{'boxName': 'box5', 'length': 120.0, 'width': 99.0, 'height': 73.0}, {'boxName': 'box5
16: [{'boxName': 'box1', 'length': 108.0, 'width': 76.0, 'height': 30.0}, {'boxName': 'box1
17: [{'boxName': 'box3', 'length': 92.0, 'width': 81.0, 'height': 55.0}, {'boxName': 'box3'
len(): 18
```

图 3-3 测试用例生成堆垛

从图中可以看出，该测试用例共生成了十八个堆垛。

后续会将箱子以每一个堆垛为单位进行装车，这样可以将三维问题转化为二维问题进行求解，进一步降低复杂度。

3.4 Box_loader(carriage_Size, boxes)

该函数主要是基于贪心算法以及随机算法对箱子进行装车。算法的总体流程图如图 3-4 所示：

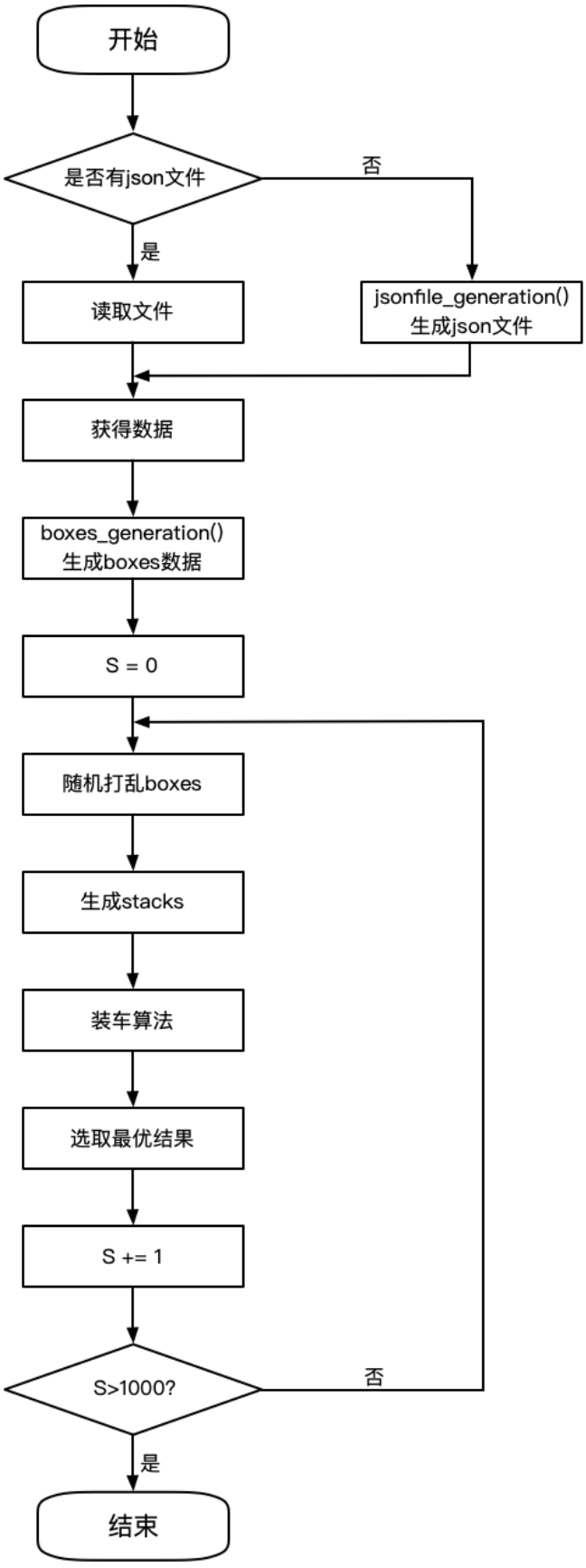
开始
是否有json文件
否
是
读取文件
jsonfile_generation()
生成json文件
获得数据
boxes_generation()
生成boxes数据
S = 0
随机打乱boxes
生成stacks
装车算法
选取最优结果
S += 1
S>1000?
否
是
结束

图 3-4 算法流程图

3.4.1 基于贪心的装车算法

在装车之前已经将箱子按照相同的长与宽堆叠成了高度不超过车高的堆垛，后续的装车算法就以堆垛为单位进行装车，这样就将三维问题转化成为了二维问题，即仅需要考虑车的长与宽和堆垛的长与宽即可。

本次算法将车体在宽度方向上分成了三排，分别为三分之一，二分之一和三分之一。装车示意图如图 3-5 所示。每一排的长度即为卡车的长度，分别记为 trucksize_r1，trucksize_r2，trucksize_r3。算法中默认从第一排开始摆放，当第一排的剩余可放长度小于堆垛长度时，再从第二排进行摆放，第三排的摆放类似。

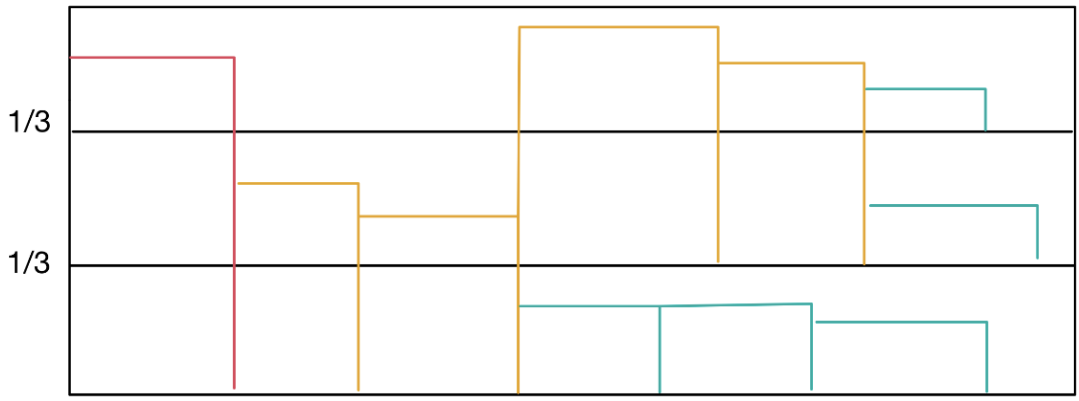

图 3-5 装车示意图

将箱体的宽度分为三种情况进行判断。

1. 堆垛宽度<=卡车宽度/3

这种情况对应图 3-5 中的绿色箱子，将其放入时他只会影响放入的那一排的剩余长度，所以放入后仅需要将放入的那一排的剩余长度减去堆垛长度即可；

2. 卡车宽度/3<=堆垛宽度<=卡车宽度*2/3

这种情况对应于图 3-5 中的黄色箱子，从图中可以看出将其放入后，它不仅会影响放入的那一排的剩余宽度，还会影响放入后上面一排的剩余宽度，所以放入此类箱子需要对两排的剩余长度都减去堆垛的长度；

3. 堆垛宽度>卡车宽度*2/3

这种情况对应于图 3-5 中的红色箱子，可以看出，将其放入后，他会影响所有排的剩余宽度，所以将该类箱子放入之后，需要将三排的剩余长度都减去堆垛宽度才能正确更新。

由于每一个堆垛的长与宽都是相同的，所以同一堆垛中的所有箱子的长度坐标与宽度坐标都是相同的，高度坐标仅需要对箱子的高度进行累加即可。

3.4.2 随机算法

从图 3-5 中可以看出，堆垛的放入顺序以及堆垛的数量对最后卡车的满载率有较大影响。为了得到较好的结果，我们在加入了随机算法。如图 3-4 中所示，3.4.1 节中的装车算法会执行 1000 次，每一次装车算法执行前都会对所有的 boxes 数据进行随机打乱，之后根据打乱后的数据生成堆垛，再按照生成堆垛的顺序依次进行放入车中。

由于每一次打乱后的 boxes 数据的顺序都不同，所以依据 boxes 数据生成的堆垛数量以及顺序都有可能不同，得到车的满载率也不同。最后从这 1000 次的随机结果中选择满载率最高的作为结果。

4 实验结果和实验结果分析

本次实验首先将数据进行预处理，将实验数据分为 3 种、5 种、8 种、10 种、15 种不同种类数量的共 5 种不同类别，分别对不同类别的数据进行 Excel 格式的录入与存储、以及 Json 格式的转化和表示。数据集最终呈现在./datalab 文件夹下的五个 Json 文件，例：“3_box.json”，示为有 3 种不同类型箱子的数据。

进行实验时，分别对这 5 个数据集进行实验，实验结果在 4.1 与 4.2 节中。

4.1 单个实验结果分析

由于车厢大小受限，箱子大小总和过大，考虑到一个车厢可能不能完全放下所有箱子，所以实验代码中引入了共 3 个车厢，在第一个车厢中已经装满且仍然有多的箱子未装入时，就将其装入第 2 个车厢乃至第 3 个车厢，实验完成之后，同时输出这三个车厢的满载率，“所有车的平均满载率”为三个车厢满载率的均值，由于后面两个车厢因为箱子已装完而出现的车厢空余，所以后面的两个车厢的满载率会拉低整体满载率，所以本次实验分析以第一个车厢的满载率作为最终满载率进行分析。如：对“5_box.json”数据集进行运行，可得结果如下：

```
一共需要车： 2 辆
第 1 辆车的满载率： 0.7410183977065845
第 2 辆车的满载率： 0.24722944324321816
所有车的平均满载率： 0.49412392047490133
最终满载率为： 0.7410183977065845
```

图 4-1 输出示例

如图 4-1 可知，这是对 5 种不同箱子装箱实验的第“E2-1”数据集的一个实验结果输出，“所有车的平均满载率”意为由于箱子数量过多，需要两个车厢进行完全装箱，但由于箱子数量不足以完全放置第二个车厢的体积，所以导致第二个车厢有大量空缺，“最终满载率”即为第一个车厢能够装的满载率，也即为本次实验想要解决求得的满载率结果。

表 4-1“5_box.json”数据集的满载率统计表

数据集编号	E2-1	E2-2	E2-3	E2-4	E2-5
满载率	0.7411±0.2	0.7426±0.2	0.7042±0.2	0.7495±0.2	0.7259±0.2

对于 5 种不同箱子的装箱问题，可以结果最高的满载率能达到接近 75%，结果最低的满载率可以达到 70.42%，说明本算法的结果完成得较好。

4.2 多个实验结果分析

接下来对 3 种、5 种、8 种、10 种、15 种不同种类数量的共 5 种不同类别的数据集进行结果展示：

表 4-2 不同种类箱子测试用例结果

数据集编号 / **不同种类满载率**	E-1	E-2	E-3	E-4	E-5
3 种	0.7279±**0.2**	0.7586±**0.2**	0.7361±**0.2**	0.6510±**0.2**	--
5 种	**0.7411±0.2**	**0.7426±0.2**	**0.7042±0.2**	**0.7495±0.2**	**0.7259±0.2**
8 种	0.8004±**0.2**	0.6232±**0.2**	0.7490±**0.2**	0.7410±**0.2**	0.6241±**0.2**
10 种	0.7261±**0.2**	0.6384±**0.2**	0.6982±**0.2**	0.6283±**0.2**	0.7433±**0.2**
15 种	0.7618±**0.2**	0.7140±**0.2**	0.6728±**0.2**	0.6377±**0.2**	0.6544±**0.2**

可以看出，在整个满载率数据当中，取得最好结果的达到了 **80.04%**，超过了 80%，最低满载率为 62.32%，超过了 60%。说明本算法完成的装箱满载率普遍较高，较好地解决了这个装箱问题。

根据每个种类的平均满载率可以画的如下图 2，可以从图表趋势线看出，种类数量越多，平均满载率呈现逐渐降低的趋势，这是符合实际逻辑的，因为种类越多，不可控因素越多，面积规划的不适配率越大，从而导致满载率降低。

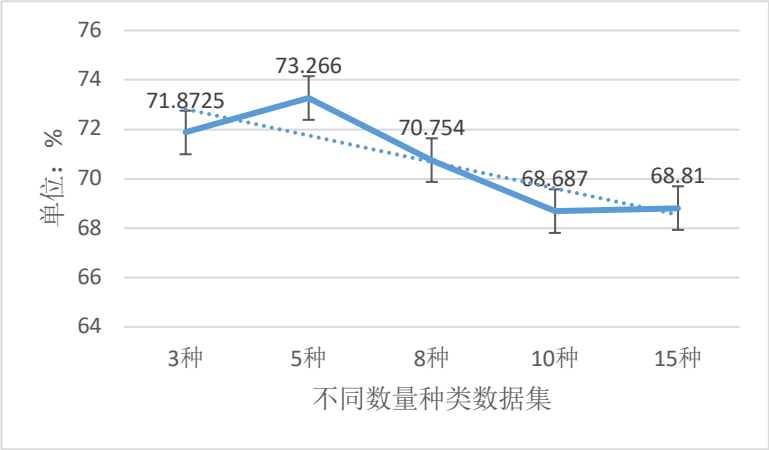

图 4-2 平均满载率走势图

5 实验总结

优点：

1. 算法的执行速度比较快，可以看出针对不同箱子的种类中的每一个测试用例，都可以在较短的时间内得到实验结果；

2. 将算法由三维问题简化为了二维问题，所以算法的复杂度有所降低；

3. 本次提供的测试用例中，长与宽相同的箱子的高也是相同的。后续如果使用高不同的箱子，本算法同样可以使用，所以具备可移植性。

缺点：

1. 从测试结果可以看出，随着箱子种类的增加，满载率呈现出下降的趋势，这是由于箱子的种类越多，箱子的宽度也就越多样，所以将宽度分成三排可能不够，需要将宽度分成更多的排数才可以得到更好的满载率；

2. 在形成堆垛的时候，本次实验将底面积相同的箱子堆放到了一起形成堆垛，实际上如果想要提高摆放的排列可能性数量，可以考虑将底面积越小的装在底面积更大的上方这种情况，提高装箱的摆放可能性数量，有可能会出现满载率更高的摆放情况；

3. 本次实验没有考虑箱子的 3 个不同面对应的 3 种不同的摆放状态，若增加摆放状态的选择性，则会增加不同底面积的堆垛，提高装箱的摆放可能性数量，

从而有可能出现满载率更高的摆放情况；

4. 算法对于堆垛放入的顺序的依赖性较大，如果面对的箱子的种类和数量较为庞大的情况，可能需要更多的随机次数才可以得到最优的结果，可能会增加算法所需要的时间。