哈 尔 滨 理 工 大 学 理 学 院

专　业　报　告

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称 | 背包问题及其扩展 | | 总 评 成 绩 |
| 题目 | 二维装箱问题 | |  |
| 姓名 | 周荣涛 | |
| 班级 | 信科18-1 | 学号 | 1807050129 |

一、背景

**1.1 什么是装箱问题**

装箱问题是复杂的离散组合最优化问题。所谓组合优化,是指在离散的、有限的数学结构上,寻找一个满足给定条件,并使其目标函数值达到最大或最小的解。一般来说,组合优化问题通常带有大量的局部极值点,往往是不可微的、不连续的、多维的、有约束条件的、高度非线性的NP完全问题。装箱问题也不例外,同许多组合最优化问题,如旅行商问题、图的划分问题等一样属于NP一HARD问题。经典的装箱问题要求把一定数量的物品放入容量相同的一些箱子中,使得每个箱子中的物品大小之和不超过箱子容量并使所用的箱子数目最少。

**1.2 装箱问题的历史由来**

**从20世纪70年代初开始,装箱问题就引起了广泛的探讨和研究。然而装箱问题可以追溯到1831年高斯(Gauss)开始研究布局问题，因为装箱问题和布局问题本质上是一样的,到现在已有百余年的历史。虽然经过几代人的努力,但迄今尚无成熟的理论和有效的数值计算方法。由于目前NP完全问题不存在有效时间内求得精确解的算法,装箱问题的求解极为困难,因此,从70~80年代开始,陆续提出的装箱算法都是各种近似算法,如下次适应、首次适应、降序下次适应和调和算法等。**

**1.3 装箱问题的应用**

**装箱问题广泛存在于工业生产,包括服装行业的面料裁剪、运输行业的集装箱货物装载、加工行业的板材型材下料、印刷行业的排样和现实生活中包装、整理物件等。在计算机科学中,多处理器任务调度、资源分配、文件分配、内存管理等底层操作均是装箱问题的实际应用,甚至还出现在一些棋盘形、方块形的数学智力游戏中。装箱问题的研究文献分布面很广,在运筹学、计算机辅助设计、计算机图形学、人工智能、图像处理、大规模集成电路逻辑布线设计、计算机应用科学等诸多领域都有装箱问题最新的研究动态和成果出现,从这个角度来讲,布局问题涉及到了工业生产的方方面面,也足以证明了装箱问题的应用前景日趋广泛而重要。**

**1.4 装箱问题的分类**

**一维装箱问题；二维装箱问题；三维装箱问题。**

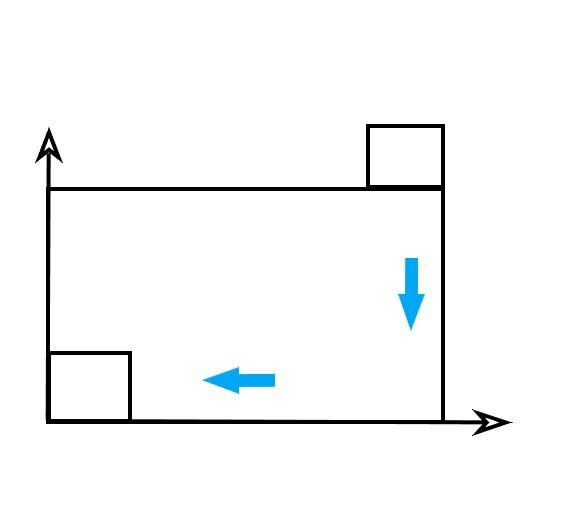
二、求解装箱问题的常见方法

**求解装箱问题的解决办法主要有：数学规划法；构造法；数值优化方法；现代优化方法等等。**

三、求解算法

3.1 本文求解算法—改进BL算法

本文在BL算法的基础上，对该算法提出了改进来求解二维装箱问题。

BL算法要求当前要装入的矩形物体要从容器的右上角进入，如下图所示。

BL算法示意图

该算法要尽可能的到达容器的底部同时尽可能的快到达容器的左侧。如果用R来表示每一个放置的物品，用序号1，2，3等等来表示物体的排列次序，则算法可以描述如下：

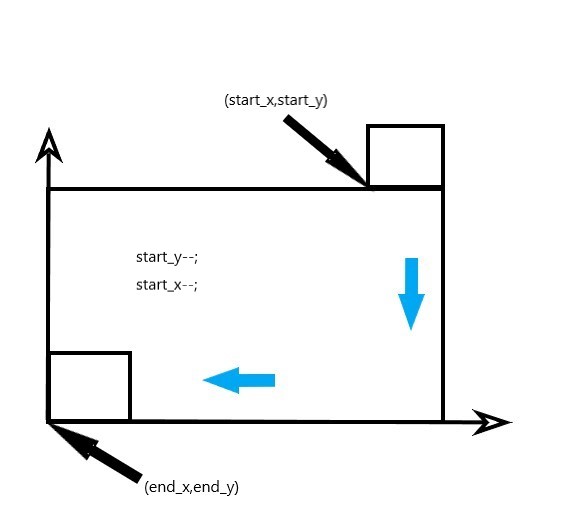
放置第i个物品到容器的左下角：移动该物品，从容器的右上角开始，尽可能快的到达底部，到达底部后尽可能快的到达左侧，同时选取底部最低的可能到达的位置。

实际应用的过程中发现，当任意序列的矩形物体装箱时，BL算法虽然运行速度比较快，但是得到的装箱结果往往不能令人满意，于是对该算法提出了下列的改进。

\*考虑到实际装箱时，先装体积和重量比较大的物体，后装体积重量比较小的物体，首先对待装物品进行排序，再按照BL算法进行计算。这样做的好处是随着矩形的不断装入，容器的空间变得越来越小且回会产生很多的碎块，如果比较小的物品最后装入，将增加填充碎块的可能性，有利于提高容器的面积利用率。

四、代码实现、测试结果及分析

代码实现过程中每个矩形的结构表示是通过左上角和右下角的两个点来进行表示的。再比较物品的大小时，是通过比较物品的面积来的。

箱子的表示则是通过二维数组来进行表示，并将箱子的左下角作为坐标轴的原点。例如既Bin[0][0]表示箱子的左下角也表示坐标轴的原点。通过改进的BL算法进行计算后，当物品不能再向箱子的左下角进行平移时，对在该物品区域内的点Bin[i][j]进行标记。于是下个物品平移时不能移动到已经标记了的区域。最终是通过返回物品最后在箱中的坐标来表示装箱后的位置，这里我是通过物品在箱中坐标的左下角和右上角来表示物品的位置。

代码思想示意图

测试代码：

package BACKPACK;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.InputStreamReader;

import java.util.LinkedList;

import java.util.List;

import java.util.Stack;

class Item{

//物品的尺寸

public int Point\_a1;

public int Point\_b1;//左上角的点

public int Point\_a2;

public int Point\_b2;//右下角的点

boolean flag1;//标记该物品是否装入箱中

Item(){

}

}

class Bin{

//装箱的尺寸

public int length\_x;

public int length\_y;

public boolean flag2;//判断箱内一点是否装入物品

public Bin(int x,int y){

this.length\_x = x;

this.length\_y = y;

}

Bin(){

}

}

public class BinPacking {

public Stack<Item> stack = new Stack();//记录装入箱中的物品

//读取文件中物品的信息--通过坐标来表示物品尺寸

public Item[] Read\_items()throws Exception {

File file = new File("C:\\Users\\Z.R.T\\Desktop\\data\_items.txt");

InputStreamReader inputStreamReader = new InputStreamReader(new FileInputStream(file));

BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

List<Integer> list = new LinkedList();

String str = null;

while ((str = bufferedReader.readLine()) != null) {

String[] s = str.split(",");

for (String value : s) {

int temp = Integer.parseInt(value);

list.add(temp);

}

}

int len = list.size();

Item[] items = new Item[len/4];

for (int i = 0;i < len/4;i++){

items[i] = new Item();

int index = i\*4;

items[i].Point\_a1 = list.get(index);//左上角横坐标

items[i].Point\_b1 = list.get(index+1);//左上角纵坐标

items[i].Point\_a2 = list.get(index+2);//右下角横坐标

items[i].Point\_b2 = list.get(index+3);//右下角纵坐标

}

for (int i = 0;i < items.length;i++){

items[i].flag1 = false;//标记物品均没有放入箱中

}

return items;

}

//计算物品的面积

public double Size(Item item1){

double a = Math.abs(item1.Point\_a1 - item1.Point\_a2);

double b = Math.abs(item1.Point\_b1 - item1.Point\_b2);

return a\*b;

}

//改进BL算法实现

public void BL(Item[] items,Bin[][] bins,Bin bin){

//对物品尺寸行进排序

BinPacking bp = new BinPacking();

int index = 0;

for (int i = 0;i < items.length;i++){

if (bp.Size(items[i]) <= bp.Size(items[index])){

index = i;

}

}

for (int i = 0;i < items.length;i++){

if (bp.Size(items[i]) >= bp.Size(items[index]) && !items[i].flag1)

index = i;//表示尺寸最大物品的标记

}

//计算物品的高度和宽度

int hight = Math.abs((items[index].Point\_b1 - items[index].Point\_b2));//物品高度

int wide = Math.abs((items[index].Point\_a1 - items[index].Point\_a2));//物品宽度

//将物品平移到箱子的右上角，然后选取物品左下角作为装入起点

int start\_x = bin.length\_x - wide-1;

int start\_y = bin.length\_y -1;

//向箱子左下角移动

//先向底部移动

while (start\_y > 0){

start\_y--;

if (!bins[start\_x ][start\_y ].flag2){

start\_y = start\_y+1;

break;

}

}

//再向左侧移动

while (start\_x > 0){

start\_x--;

if (!bins[start\_x ][start\_y ].flag2){

start\_x = start\_x+1;

break;

}

}

//物品移动到左下角的坐标

int end\_x = start\_x;

int end\_y = start\_y;

if ((end\_x + wide) >= bin.length\_x || (end\_y + hight) >= bin.length\_y){

return;

}else {

//标记物品放入区域的点为已选中

for (int i = end\_x;i < (end\_x+wide);i++){

for (int j = end\_y; j < (end\_y+hight);j++){

bins[i][j].flag2 = false;//标记该区域选中了物品

}

}

items[index].flag1 = true;//标记该物品已经装入箱中

stack.add(items[index]);//将选中物品加入栈中

System.out.println("装入物品的编号：" + index);

System.out.println("装入箱后矩形的坐标：");

System.out.println("左下角" + "(" + end\_x + "," + end\_y + ")");

System.out.println("右上角" + "(" + (end\_x + wide) + "," + (end\_y + hight) + ")");

System.out.println("---------");

}

}

//测试类

public static void main(String[] args) throws Exception{

//创建空箱

Bin bin = new Bin(15,15);

Bin[][] axis = new Bin[bin.length\_x][bin.length\_y];

for (int i = 0;i < bin.length\_x;i++){

for (int j = 0;j < bin.length\_y;j++){

axis[i][j] = new Bin();

axis[i][j].flag2 = true;//箱中没有装任何物品

}

}

//读取物品信息

BinPacking bp = new BinPacking();

Item[] items = bp.Read\_items();

//开始装箱

for (int i = 0;i < items.length;i++){

bp.BL(items,axis,bin);

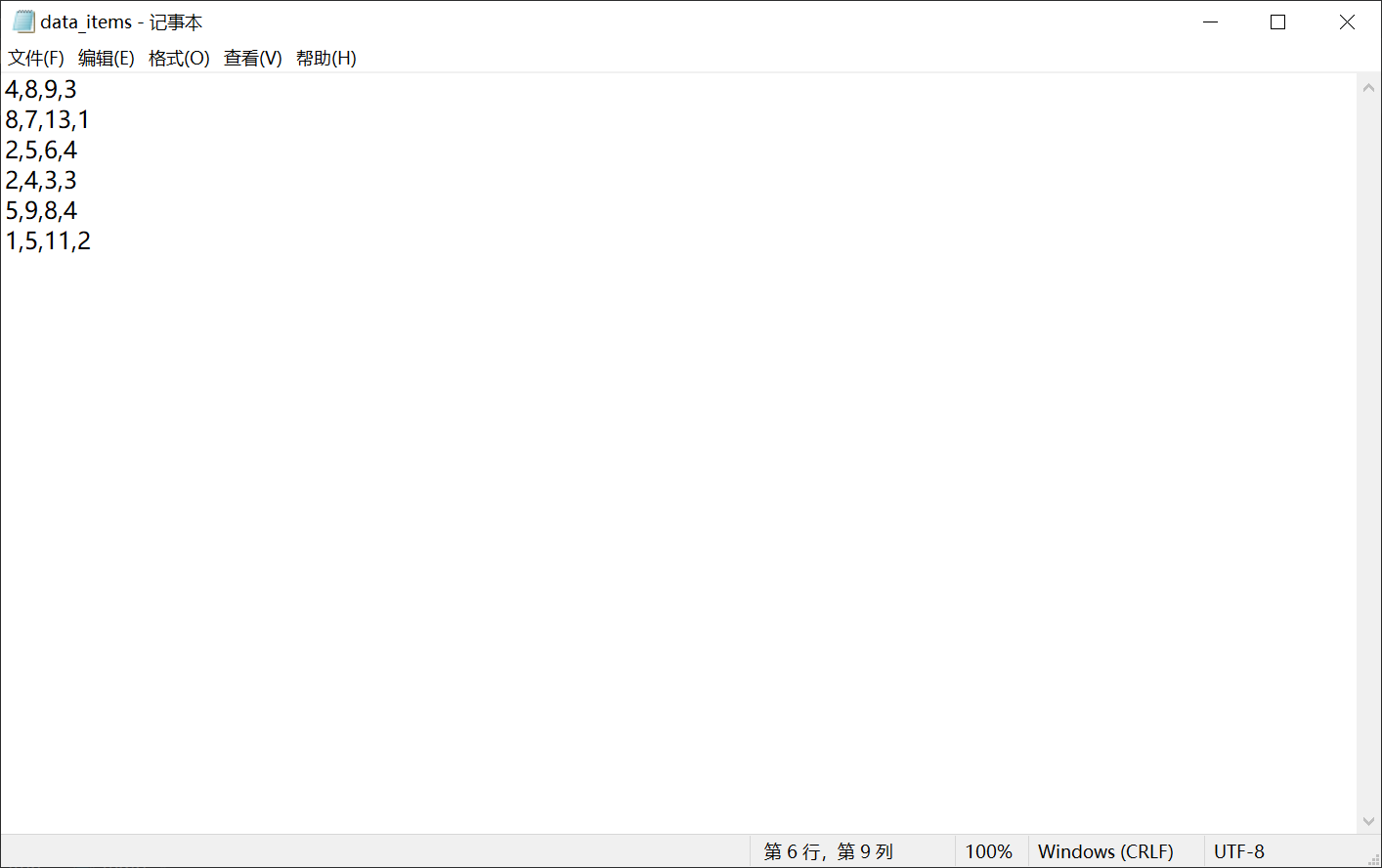
}

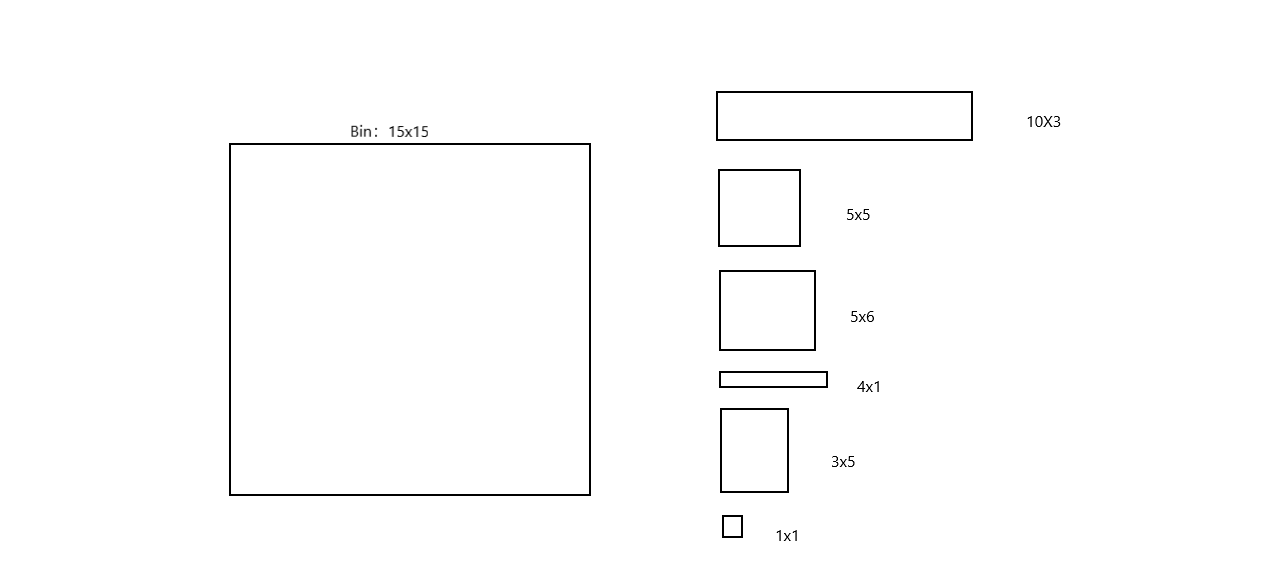
}

}

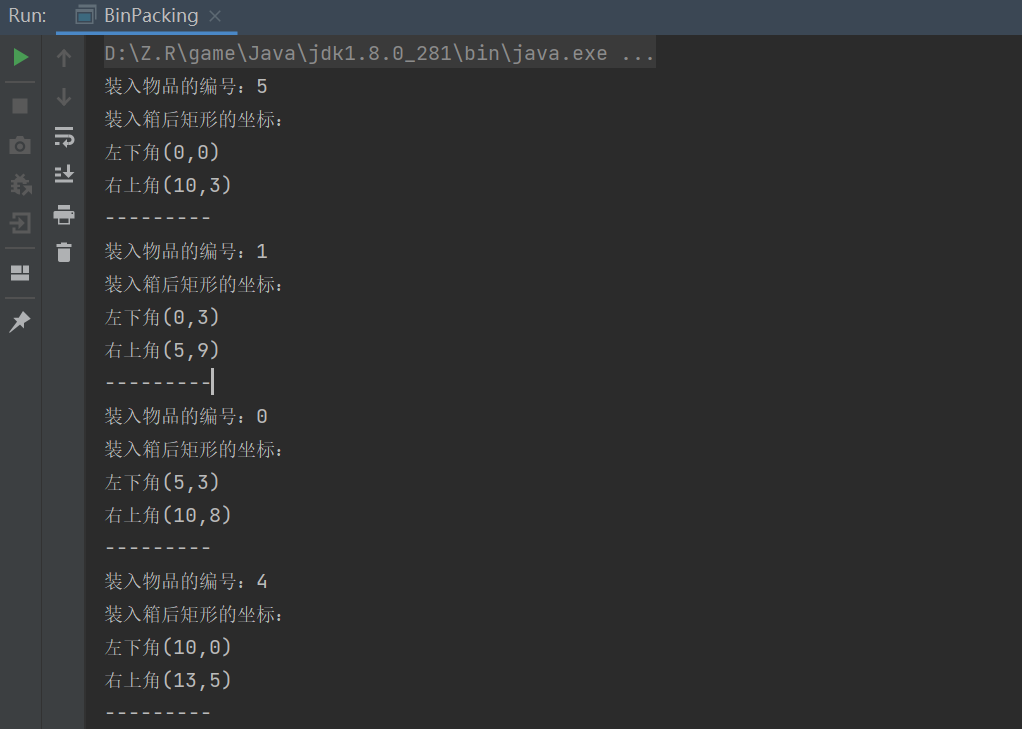
测试数据如下图所示：

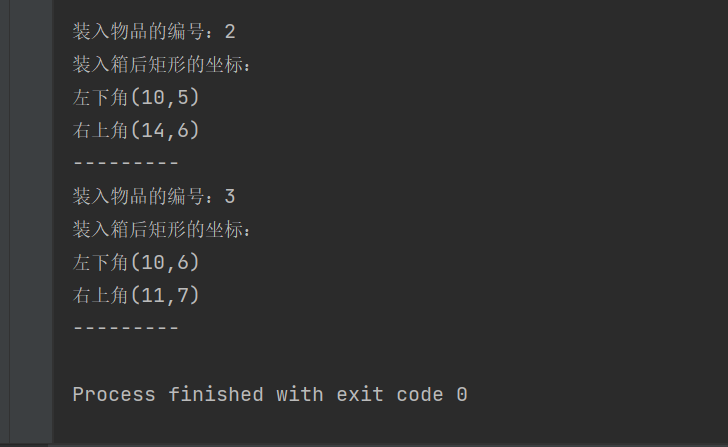
数据通过坐标表示：一行数据前两位表示左上角坐标，后两位表示右下角坐标



几何示意图如下：

运行结果如下：



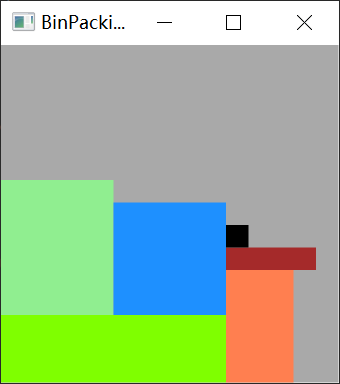


通过JavaFx绘制图像：

将计算出来的坐标进行坐标变换：通过左下角(x,y)和右上角(X,Y)的坐标可以计算出左上角的坐标为(x,Y),通过坐标变换后左上角的坐标为(225-15x,225-15Y)。这里是将箱子高宽均扩大了15倍。

绘制代码：

import javafx.application.Application;import javafx.scene.Parent;import javafx.scene.Scene;import javafx.scene.layout.Pane;import javafx.scene.paint.Color;import javafx.scene.shape.Rectangle;import javafx.stage.Stage;public class temp extends Application { private Rectangle rectA; private Rectangle rectB; private Rectangle rectC; private Rectangle rectD; private Rectangle rectE; private Rectangle rectF; private Rectangle rectG; public Parent createContent() { rectA = new Rectangle(150, 45, Color.CHARTREUSE); rectA.setX(0); rectA.setY(180); rectB = new Rectangle(75, 90, Color.LIGHTGREEN); rectB.setX(0); rectB.setY(90); rectC = new Rectangle(75, 75, Color.DODGERBLUE); rectC.setX(75); rectC.setY(105); rectD = new Rectangle(45,75,Color.CORAL); rectD.setX(150); rectD.setY(150); rectE = new Rectangle(60,15,Color.BROWN); rectE.setX(150); rectE.setY(135); rectF = new Rectangle(15,15,Color.BLACK); rectF.setX(150); rectF.setY(120); rectG = new Rectangle(225,225,Color.BURLYWOOD); rectG.setX(0); rectG.setY(0); Pane root = new Pane(rectG,rectA, rectB, rectC,rectD,rectE,rectF); root.setPrefSize(225, 225); root.setMinSize(225, 225); root.setMaxSize(225, 225); return root; } @Override public void start(Stage primaryStage) throws Exception { primaryStage.setTitle("BinPacking"); primaryStage.setScene(new Scene(createContent())); primaryStage.show(); } public static void main(String[] args) { launch(args); }}

绘制结果如下：

当对所有物品根据面积进行排序后，在使用BL算法进行装箱时，可以知道箱中底部优先被面积大的物品填充，底部的空隙比较小。

五、实验结论

本文介绍了装箱问题并通过改进BL算法对二维装箱问题进行了求解。在装箱之前首先对物品按照面积的大小进行排序，这样就一定层度上减小了箱中底层的空隙。但这样的改进仍具有一定的缺陷，例如容易陷入局部最优解。于是可以再对BL算法做出进一步的改进--考虑矩形的适箱性由于装箱方向的不同而不同，对矩形在装入位置进行旋转操作。