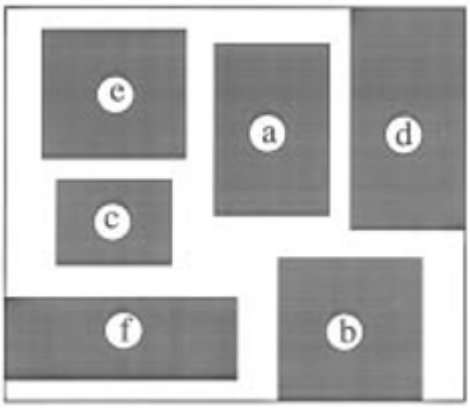
基于排列对的最小包络矩形求解算法

计71 张晨 2017011307

问题描述

最小包络矩形（rectangle packing problem）问题是指，给定若干个大小任意的小矩形，找到一个面积尽可能小的大矩形，使这些小矩形可以无重叠地被放置在这个大矩形中。



基础算法

通过将矩形向左下、右上延伸和向左上、右下延伸，可以将每一个矩形排布方法对应于两个长度为矩形个数的排列A、B，满足若i在j的左边，则在两个排列中i都出现在j之前，若i在j的上面，则在排列A中i出现在j之前，且在排列B中，i出现在j之后。例如，上图对应的排布方法对应的排列为A='ecadfb',B='fcbead'。

因此，对于一个排列对，可以生成一系列的形如i应当在j的左边/上面的限制条件，且可以证明一个满足所有限制条件的排布方式都能满足矩形之间没有重叠。满足限制条件的最优的矩形排布方式可由以下最长路算法求出：

(1)基于所有i在j的左边的限制条件，可构造一个有向图G=(V,E)。构造方式如下：

V：源点S，以及n个点，分别代表n个小矩形。

E：对每个小矩形，连接一条S到它的长度为矩形宽度的有向边。对于每个形如i应当在j的左边的限制条件，连接一条i到j的长度为矩形i的宽度的有向边。

求S到每个点的最长路，即可得出每个小矩形左下角的点的横坐标。

(2)相似的，可以根据形如i应当在j的右边的限制条件，求出每个小矩形左下角的点的纵坐标，从而确定每个矩形的位置。

因为这两个有向图都是拓扑图，所以可以通过拓扑排序+动态规划的方式完成最长路的求解。

于是，对于一组小矩形，只要能寻找到最优的排列对，即可寻找到最优的矩形排布方式。寻找最优排列对可以通过模拟退火的方法进行。

改进算法

对于一个排列对，最优矩形排布也可以转化为最大权公共子串问题。

带权字符串指一个定义了字符集中每个字符的权值$W(S_i)$的字符串。两个带权字符串的最大权公共子串指这两个字符串的所有公共子串中，字符的权值之和最大的那一个公共子串。

与基础算法类似，首先处理所有i在j的左边的限制条件。将元素i的权值定义为第i个矩形的宽度。对于第i个矩形，排列对(A,B)可改写为(A_1iA_2, B_1iB_2)，可以证明，矩形i左下角的横坐标为A_1与B_1的最大权公共子串的权值。

对于i在j的上边的限制条件，将元素i的权值定义为第i个矩形的高度。令A^R表示将字符串A翻转得到的字符串。对于第i个矩形，将排列对(A, B^R)改写为(A_1iA_2, B_1iB_2)，则也可以证明，矩形i左下角的纵坐标为A_1与B_1的最大权公共子串的权值。

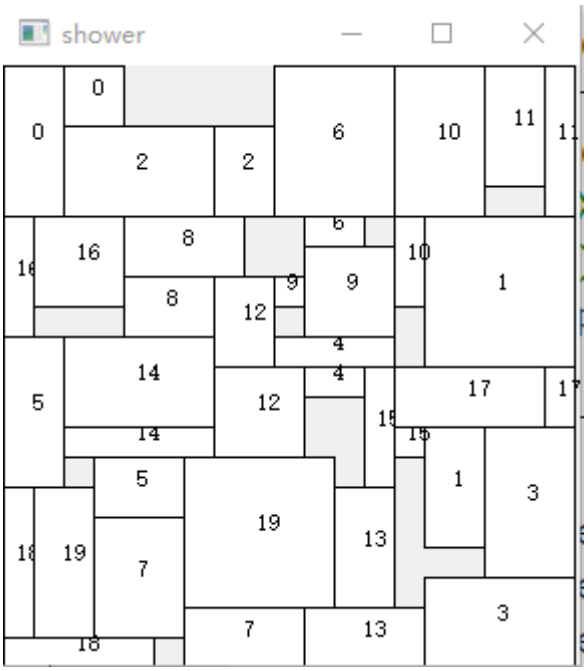
最大权公共子串问题可以通过动态规划求解，其时间复杂度为$O(n^2)$，也可以使用论文中提出的双向链表+二叉树的方法求解，时间复杂度为$O(n\ log\ logn)$。

问题拓展

使用同样的算法，也可以求解其他矩形排布问题，如可以将小矩形每2个一组分为若干个矩形对，求解一个包络矩形面积与所有矩形对的两个矩形的距离之和（两个矩形的距离指这两个矩形中心点的距离）都比较小的矩形排布方式。更形式化的，令S表示包络矩形面积，L表示距离之和，该算法可以求解使$\alpha * S + (1-\alpha) * L$值最小的排布方式。

结果可视化

由于需要观察结果来评估运行结果，从而调整模拟退火中初始温度、终止温度、降温速度等参数，结果需要以一种比较直观的方式表现出来。因此，我利用QT编写了一个较为简单的可视化程序，从文件中读入矩形排布方式，然后绘制矩形。每个矩形的中心有一个编号，编号相同的为一个矩形对。



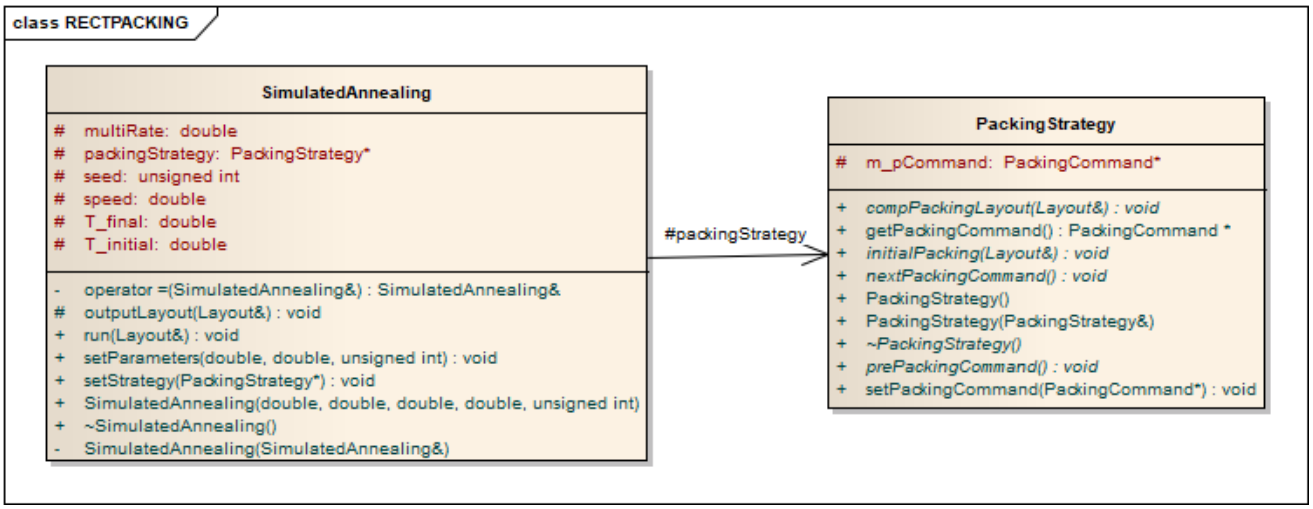
使用的设计模式

策略模式

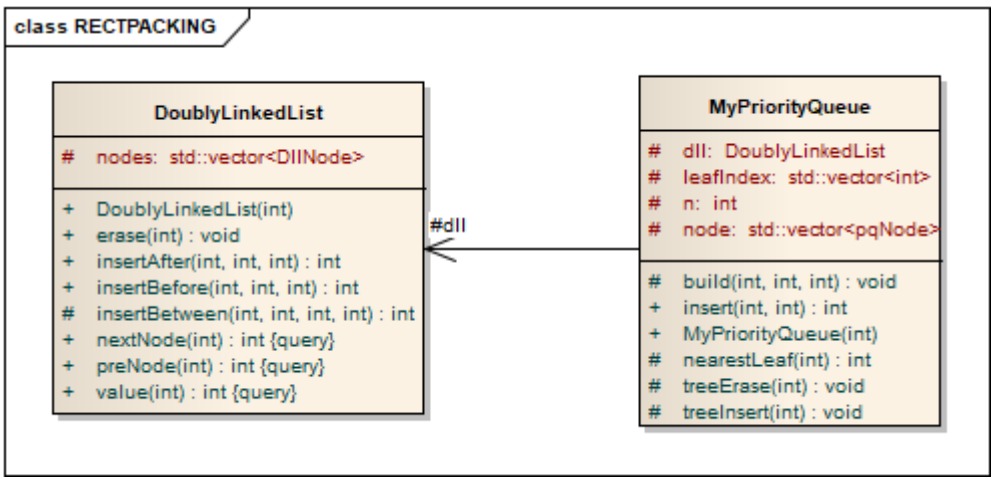
计算一个排列对的最优排布有基于最短路和基于最大权公共子串的两种算法。为对比这两种算法的运行时间，我采取了策略模式，将这两种算法视为模拟退火时计算当前状态权值的两种策略，编写了两个有同一基类的策略类，实现了这两个算法间的方便的相互替换。

核心部分UML图

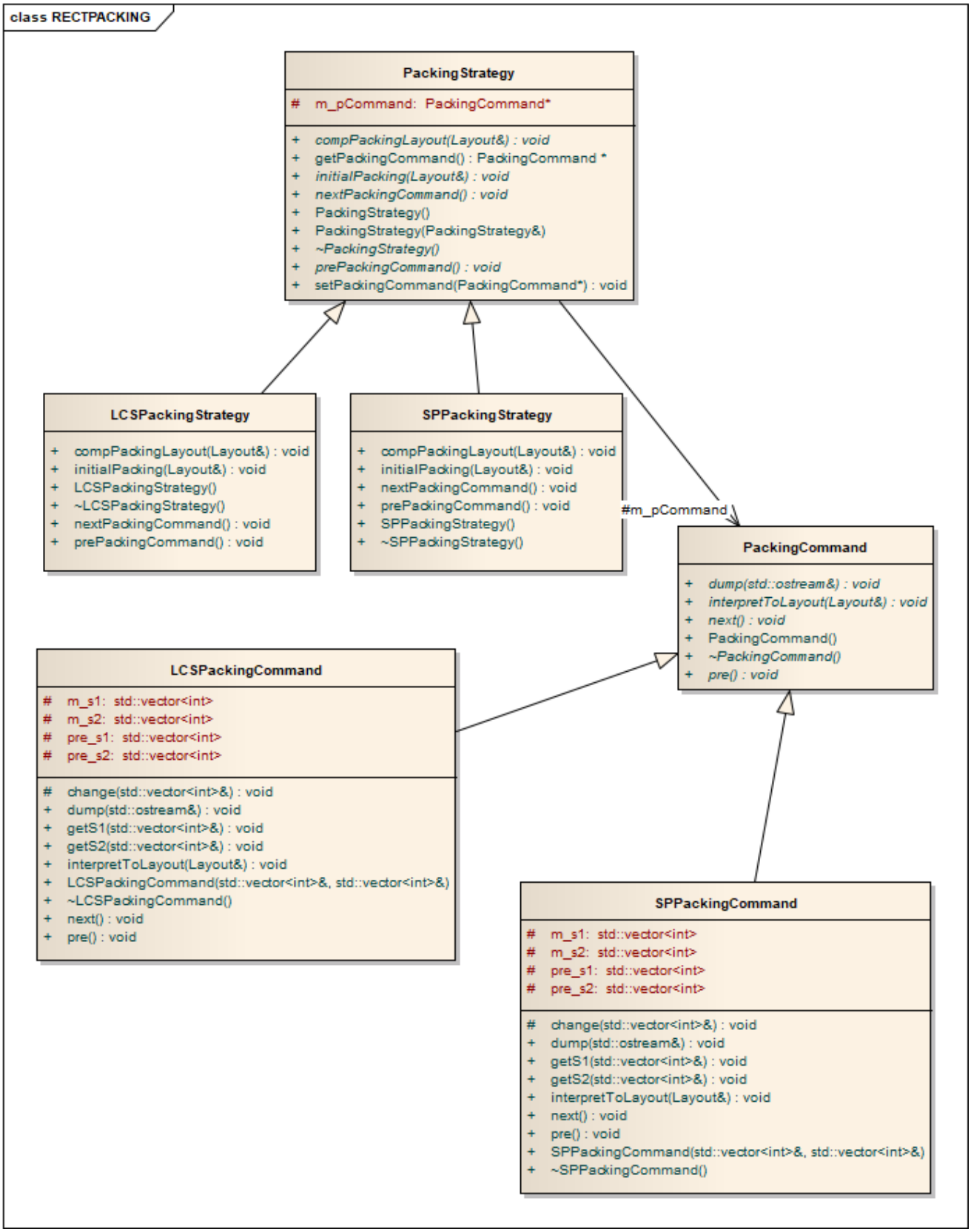
- `SimulatedAnnealing`



- `DoublyLinkedList` & `MyPriorityQueue`



- `PackingStrategy` & `PackingCommand`

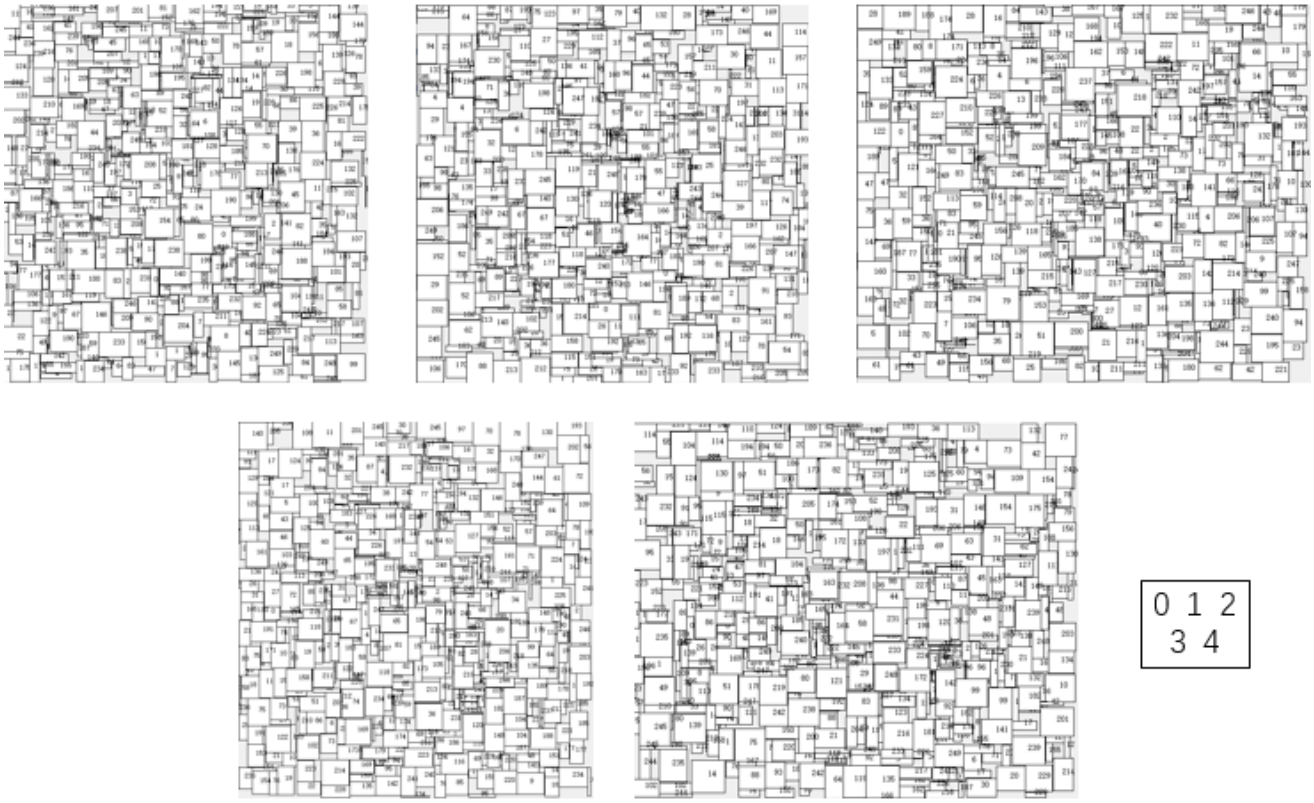


排布效果

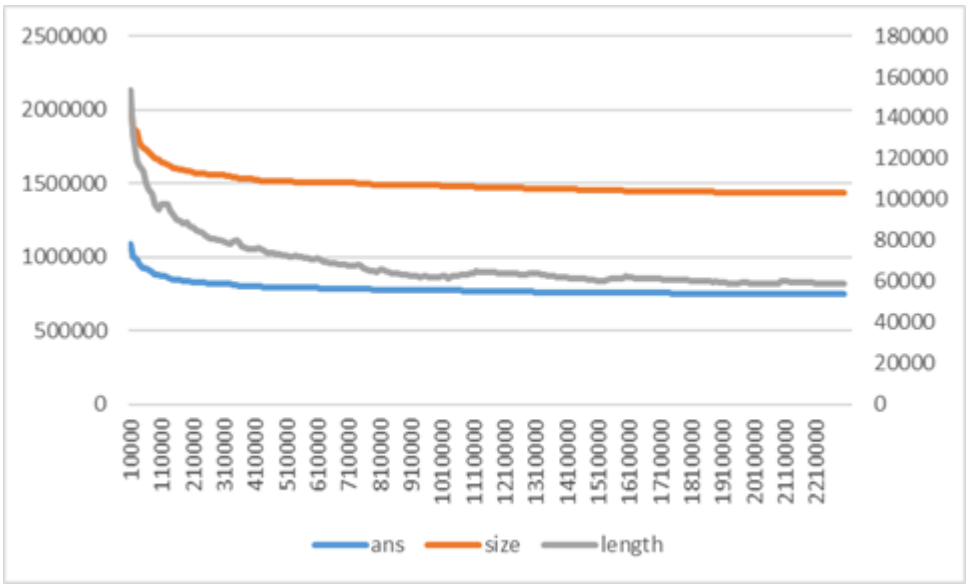
进行了5组有250个矩形对,共计500个矩形的排布实验.排布实验中，设α的值为0.5

矩形的产生方式为从1~100的整数中分别等概率抽取其宽度和高度。

每次排布实验中,模拟退火的参数都为:初始温度1,终止温度0.1,降温速度0.999999. 单次模拟退火会产生2302584个排列对，五组的平均运行时间为17.6分钟。这组参数在5组测试数据中都取得了不错的效果。



第一组的总代价、面积、距离和随轮数的变化如下：（总代价、面积依据左坐标轴，距离和依据右坐标轴）



五组测试数据的总面积、总线长、加权平均值(α*面积+$(1-\alpha)$*线长)、空间利用率(矩形面积和/总面积)、平均线长(总线长/矩形对数)如下表所示。

测试数据编号	总面积	总线长	加权平均值	空间利用率	平均线长
0	$1.37 * 10^6$	$6.02 * 10^4$	$7.16 * 10^5$	86.5%	248
1	$1.43 * 10^6$	$5.87 * 10^4$	$7.47 * 10^5$	86.9%	235
2	$1.43 * 10^6$	$6.27 * 10^4$	$7.46 * 10^5$	87.9%	251
3	$1.50 * 10^6$	$6.43 * 10^4$	$7.83 * 10^5$	87.7%	257
4	$1.43 * 10^6$	$6.19 * 10^4$	$7.45 * 10^5$	87.2%	248

两算法运行时间对比

时间的测量方式为：对于每种矩形个数，随机生成5组对应个数的矩形。对每组矩形，分别调用1000次这两种算法，统计两种算法在5组测试数据中的运行总时间。速度比的计算公式为最短路运行时间/公共子串运行时间

矩形个数	最短路	公共子串	速度比
6	0.168	0.011	14.9
10	0.338	0.014	25.0
20	1.024	0.024	43.0
50	5.365	0.048	111.4
100	19.535	0.094	208.3
200	75.486	0.187	404.0
500	453.037	0.476	950.8

仍存在的不足之处

1. 这组模拟退火参数仅在“矩形的产生方式为从1~100的整数中分别等概率抽取其宽度和高度”这种矩形生成方式中有较好表现，在其他矩形产生方式，如矩形边长在1~5之间随机，表现便不太理想。在实现模拟退火时，我预留了一个用以针对矩形大小进行调整的参数，但没有深入研究这个参数的具体调整方式。我也没有研究在矩形个数为其他值时，应如何调整降温速度以达到效率与效果的统一。
2. 受电脑计算能力限制，没有测量基于最短路的算法在有500个矩形时的运行时间。
3. 由于基于排列对的求解算法会尽可能将矩形向一个角聚拢，在很多情况下，尤其是矩形数量较少的情况下，它并不能产生最优解。例如，在输入数据为如下图的两个矩形时，得到的解为左图，而最优解实际为右图。