CornerBlockList 说明文档

何钦尧 邹昊 庄天翼

一、

问题说明：

给定 500 个大小的矩形，两两配对。将它们摆放在二维平面中，在互不重叠的情况下，要求包含这些矩形的最小矩形的面积 A 和配对矩形中心的曼哈顿距离和 L 的加权和 $\alpha A+(1-\alpha)L$ 最小。

二、

实现算法：

这是一个 NP-Hard 问题，只能用搜索的方法解决。为了搜索更加高效，需要有一种有效的表示排布方案的方法。最后用模拟退火的方法搜索求得较优解。

表示排布方案的方法选择 CornerBlockList.

在本实验的实现中，CornerBlockList 包含 4 个部分，S、L、T 以及 R。

S 长度为 N（N 为矩形个数）包含了所有矩形的编号，代表了矩形的一个在构造排布中的排列顺序。

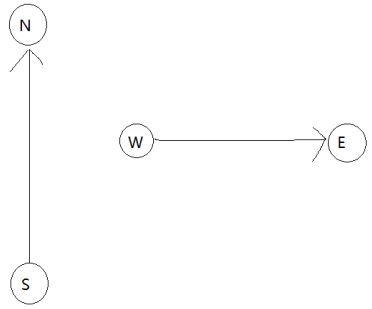
L 包含了一个长度为 N-1 的 0-1 串，表示各个矩形在加入排布的方法（水平加入或者竖直加入）

T 包含了一个 0-1 串，可分为 N-1 段，每段以 0 结尾，在 0 之前有长度为 P-1（P-1 可为 0）的 1 串，表示这个矩形在加入的时候（水平加入或竖直加入）覆盖了 P 个矩形。

R 表示了各个矩形的摆放状态（是否旋转 90 度，旋转 180 度与不旋转一样，旋转 270 度与旋转 90 度一样）

下面讨论给定 CornerBlockList 下的方案生成。

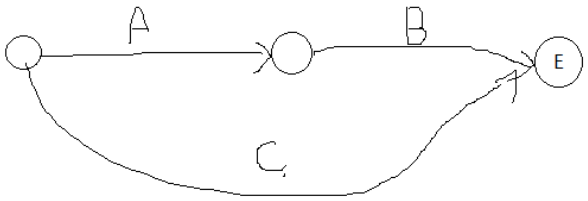
设置四个点表示最后包含所有矩形的大矩形的上下左右边沿，记为 N、S、E、W。生成两张图，水平方向图和竖直方向图。水平方向图中 W 为起点、E 为终点，竖直方向图中 S 为起点，N 为终点。初始摆放一个矩形（即 S 串中第一个矩形）。定义一个矩形对应了矩阵竖直边、矩阵水平边，边权分别为高度和宽度。在 N、S 之间加入一条该矩阵的矩阵竖直边，E、W 之间加入一条该矩阵矩阵水平边。如下图所示：



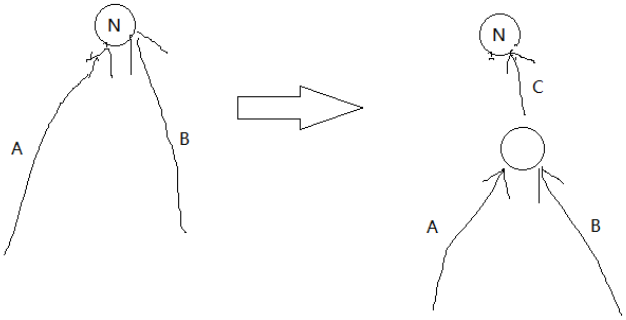
在水平方向图和竖直方向图中节点代表了矩形边沿，边代表了矩形。

随后根据 S、T、L 代表的结构生成矩形的相对摆放方案，并扩充水平方向图和

竖直方向图。
以竖直方向加入为例：
假设矩形 C 竖直方向加入，覆盖 A、B（A 左,B 右）两个矩形。
水平方向图中，A 矩形的矩阵水平边的起点与 E 之间加入 C 的矩形水平边。
如下图所示



竖直方向图上：
原本 A、B 的矩形竖直边的重点为 N，现在新加入一个节点，将 A、B 的矩形竖直边指向这个新节点。新节点与 N 之间加入 C 的矩形竖直边。



最后对于矩形 A，A 所在摆放的范围按如下规则确定：
1、A 的左边沿横坐标不小于 A 在水平方向图中 A 的矩形水平边的起点和 W 之间的最长路径。
2、A 的右边沿横坐标不大于 A 在水平方向图中 A 的矩形水平边的终点和 W 之间的最长路径。
3、A 的下边沿纵坐标不小于 A 在竖直方向图中 A 的矩形竖直边的起点和 S 之间的最长路径。
4、A 的上边沿纵坐标不大于 A 在竖直方向图中 A 的矩形竖直边的终点和 S 之间的最长路径

确定了每个矩形的摆放范围后，Packing 的面积也就确定了，即 S 到 N 的最长路径与 W 到 E 的最长路径的乘积。
最后要让连线和最小，每对矩形之间互不干扰。只需让每对矩形之间的中心连线最短即可。
最后面积和连线长度的加权和即为一个 CornerBlockList 的估价函数。

最后用模拟退火的方法搜索求得较优解，选定初始 CornerBlockList，每次在当前 CornerBlockList 的基础上稍加改动，得到一个新解，进行模拟退火的比较和决策。

新解的产生方法有 4 种：
1、随机交换 S 串中两个矩形的位置
2、随机选择 L 串中一个 0-1，将其取反
3、随机在 T 串中选择一个位置，随机加入 1，或去掉一个 1.
4、随机选择 R 串中一个矩形，将其旋转。
若新解不合法，在模拟退火中肯定不接受。
最后将经过模拟退火计算后的结果用 openCV 图形化展示。

三、
类、文件及接口的说明：
1.Common.h
保存了整个问题中所用到的一些参数。
RECTANGLE_NUM 矩形个数
RECTANGLE_WIDTH_MAX 矩形宽度最大值
RECTANGLE_WIDTH_MIN 矩形宽度最小值
RECTANGLE_HEIGHT_MAX 矩形高度最大值
RECTANGLE_HEIGHT_MIN 矩形高度最小值
PAIR_NUM 矩形对的数目
PARAMETER 面积和连线长度的加权和参数
INI_TEMP 模拟退火的初始温度
GOAL_TEMP 模拟退火的目标温度
START_NUM 初始一个温度下的迭代次数
DEC 温度下降的倍率
INC 迭代次数的增长倍率

2.TestCaseGenerator.h
使用了单例模式，用于产生测试数据
成员函数：
void generate(int cnt); 产生 cnt 个测试数据

3.Rectangle.h
矩形类，代表了一个矩形
成员变量：
double center_x; 矩形中心横坐标
double center_y; 矩形中心纵坐标
double width; 矩形宽度
double height; 矩形高度
Edge *V_edge; 矩形的矩形竖直边
Edge *H_edge; 矩形的矩形水平边
成员函数：
Rectangle(double w, double h) 矩形的构造参数，w 为宽度，h 为高度
Rectangle() {} 默认构造函数
double CENTER_X()、double CENTER_Y() 返回矩形中心的横坐标及纵坐标

double GET_WIDTH(bool s) 返回矩形的宽度（s 代表旋转情况）
double GET_HEIGHT(bool s) 返回矩形的高度（s 代表旋转情况）
Edge* GET_V() 返回矩形的矩形竖直边
Edge* GET_H() 返回矩形的矩形水平边
void SET_X(double x) 设置矩形的中心横坐标
void SET_Y(double y) 设置矩形的中心纵坐标

4.Node.h
代表了水平方向图和竖直方向图上的点
成员变量：
std::vector<Edge*> edges; 存储了以该点为起点的边
double dis; 代表了所在图的起点到该点的最长路径
int Go_in; 代表了该点的入度（用于拓扑排序）
成员函数：
Node() 默认构造函数
void Add(Edge* e) 加入一条以该点为起点的边

5.Edge.h
代表了水平方向图和竖直方向图上的边
成员变量：
double length; 代表了该边的长度
Node* s; 代表了该边的起点
Node* t; 代表了该边的终点
成员函数:
Edge() 默认构造函数
Edge(double len) 构造函数，长度为 len
void SetLen(double x) 设置边的长度
void SetS(Node *x) 设置边的起点
void SetT(Node *x) 设置边的终点
Node* GET_S() 获取边的起点
Node* GET_T() 获取边的终点
double GET_LEN() 返回边的长度

6.State.h
存储了 CornerBlockList 的四个串
成员变量：
vector<int> block_ids; 即 S 串
vector<bool> orientations; 即 L 串
vector<bool> uncover_rec_num 即 T 串
vector<bool> ifrotate 即 R 串（是否旋转）
成员函数：
Content() 默认构造函数
Content(const Content* c) 构造函数，拷贝 c 中的内容
~Content() 析构函数

7.CornerBlockList.h
存储了所有的矩形信息，和当前的 CornerBlockList 串以及相关的一些变量，用于整个问题的计算。

成员变量：
vector<Rectangle> rectangles 存储了所有的矩形信息
State* con; 存储了当前的 CornerBlockList 串
vector<int> HStack、vector<int> VStack 用于方案生成的两个栈，用于记录当前右边沿和上边沿的矩形编号
int block_num,pair_num; 矩形个数和矩形对数目
Node* E; 水平方向图的终点
Node* N; 竖直方向图的终点
Node* W; 水平方向图的起点
Node* S; 竖直方向图的起点

成员函数：
State* RandomChange(const State* c); 在 c 的基础上生成一个新解
bool build(const Content* c) 根据 c 对应的方案生成水平方向图和竖直方向图
void cal_longest(Node* start) 以 start 为起点计算到各个点的最长路（包括拓扑排序和计算最长路）
CornerBlockList(); 默认构造函数
CornerBlockList(const std::string& file_name); 构造函数，读取 file_name 文件中的矩形信息
void show(); 展示模拟退火后计算得到的较优解
double assess(const Content* c); 对 CornerBlockList 的 c 对于的方案进行计算得到一个估价值
void optimize(); 模拟退火优化（用到了 assess 函数）

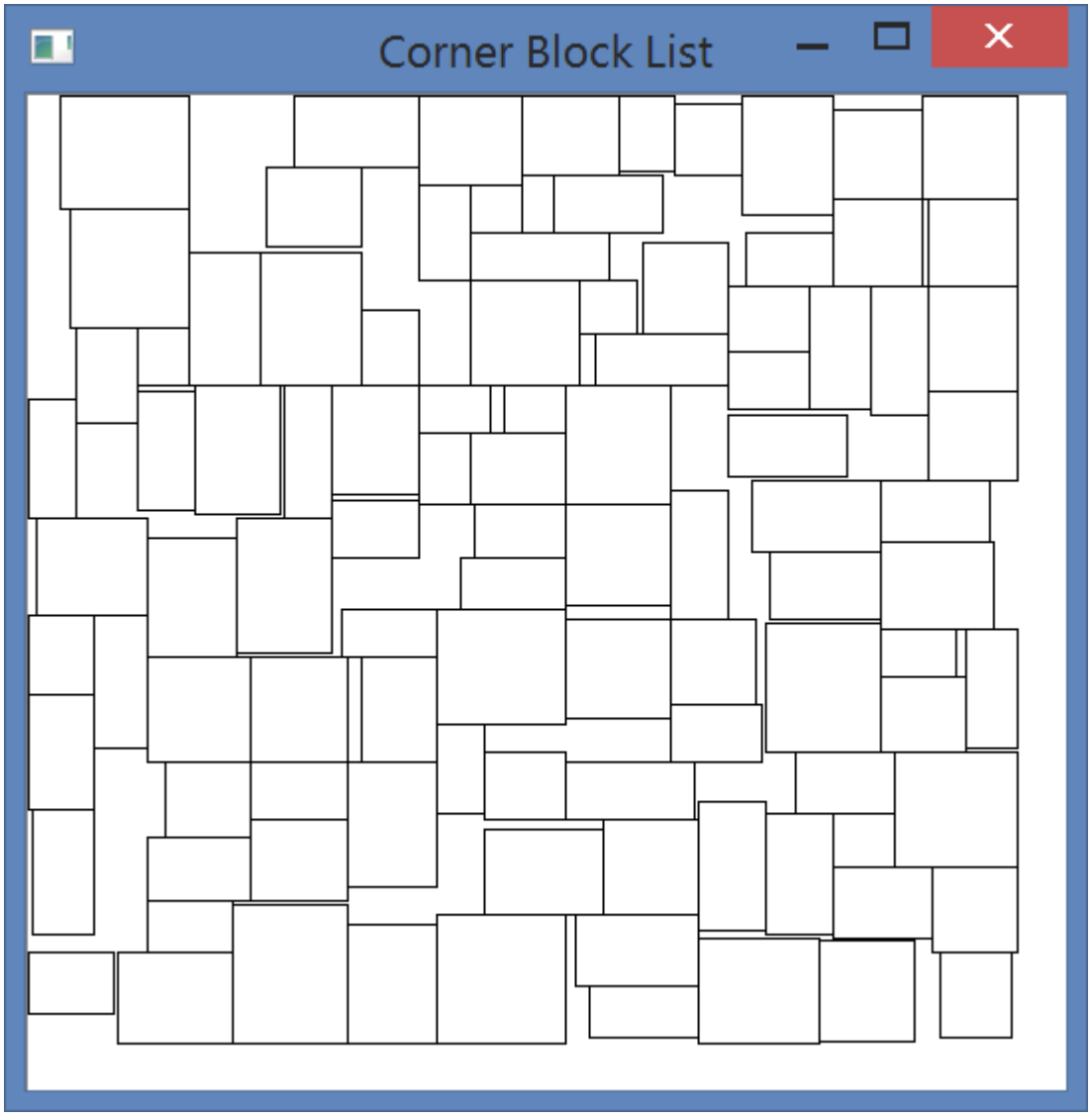
五、程序流程

```cpp
#include <iostream>
#include "common.h"
#include "TestCaseGenerator.h"
#include "CornerBlockList.h"
#include <fstream>
#include <vector>
#include <string>

using namespace std;

int main() {
    test::TestCaseGenerator *generator = test::TestCaseGenerator::Instance();

    generator -> generate(TEST_FILE_NUM);

    vector <CornerBlockList> cbls;
    for (int i = 0; i < TEST_FILE_NUM; i++)
        cbls.push_back(CornerBlockList(FILE_NAME_PREFIX + char(i+'0') + FILE_NAME_SUFFIX));
    for (int i = 0; i < TEST_FILE_NUM; i++)
        cbls[i].optimize();
    for (vector <CornerBlockList>::iterator cbl = cbls.begin();cbl!=cbls.end();cbl++)
        (*cbl).show();

    delete generator;
    return 0;
}
```

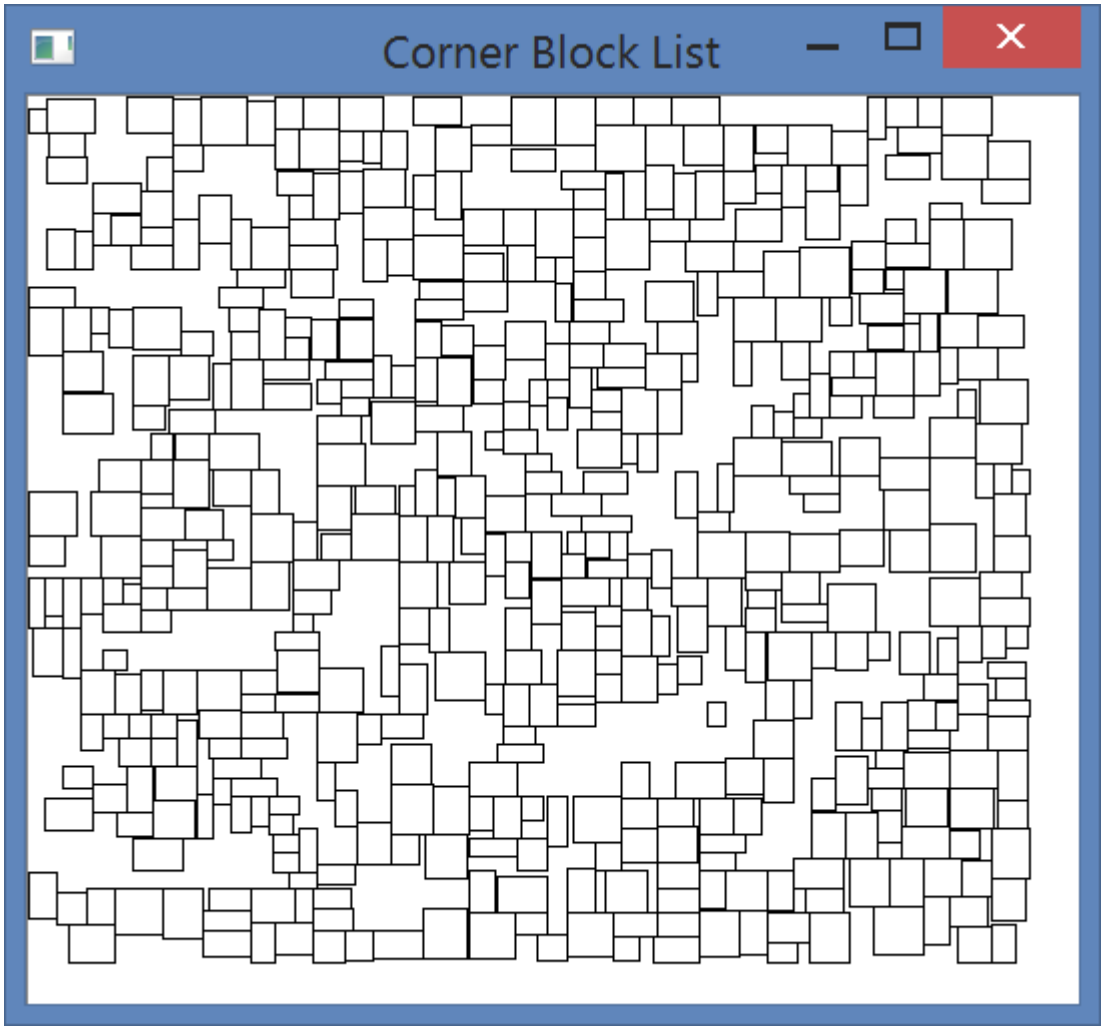
先用单例模式的 generator 产生测试数据，生成对应个数的 CornerBlockList 对象读取测试数据，存储。
每个 CornerBlockList 对象执行 optimize()函数模拟退火优化，最后用 show()函数展示结果。

六、结果展示

下图是 100 个矩形的运行效果，大约运行了 20 秒。



下图是 500 个矩形的运行效果，大约运行了 10 分钟。



可以看出，在矩形数量比较多的时候效果并不好。可能需要进一步的调整退火参数，包括初始温度，以及每个温度的迭代次数等。但是考虑到 10 分钟的运行时间已经有些长了，所以有时宁可用更短的时间来换稍微差一些的效果。