面向对象程序设计大作业

RECTILINEAR STEINTER TREE 设计文档

计 34 何钦尧 2012010548

材 43 庄天翼 2014012056

计 43 邹昊 2013011016

1. 整体设计

在 main 函数中生成若干个点，加入 RST 类中。最后的求解结果用 OpenCV 画图显示。边用黑线，原来的点用蓝色标记，三条线段的交叉口就是 Steiner Point（没有特别标出）。

使用了 Strategy 设计模式，RST 类可以使用 changeStrategy 函数来改变所使用的求解模式（L-RST 或者 Z-RST）。RSTStrategy 为抽象类，LRSTStrategy 和 ZRSTStrategy 分别为这两个的 Strategy 类。ZRST 和 LRST 类用于求解这两种不同的算法。ZRST 和 LRST 实现时候接口不同（不同人开发的），所以 Strategy 类也相当于起到了 Adapter 的作用。

此外有一些通用的工具类。包括 Point，Segment，Line。Line 定义了基类 Line 和派生类 Line_L，Line_Z 为 L 型和 Z 型 Layout。Segment 主要在外围 RST 类和 Visualizer 类中使用，表示每一个水平或者竖直的线段（已经将折线拆分）。

MST 类为求解直角最小生成树，为之后的算法做准备。LRST 和 ZRST 中都保存这个类的一个实例，用于求解直角最小生成树后从中得到所有的边。

Common.h 中定义了若干常量。

Visualizer 类起到了显示的作用。为一个单例。

2. 技术实现

下面主要讨论 LRST 和 ZRST 的算法实现。

2.1. LRST

LRST 是要对每一条边选择“L 型”或者“U 型”layout，在所有的方案当中，求出 overlap 最大的那个。由于具有 separable 的性质，所以实际上子问题可以独立，穷举搜索可以转化为在树上的动态规划过程。

因此需要首先建立这颗树。需要选择一个度为 1 的节点作为根（这是保证初始只有一条边），find_root 函数完成这一功能。然后利用 MST 中求出的边列表来递归的建立这棵树。树由 tree 和 parent 两个数组表示。Tree 中每个元素为一个 vector，表示某个节点的后继节点。Parent 顾名思义。

由于任何求出的边，都只可能在有点存在的 x 坐标或者 y 坐标的网格线上，因此将坐标离散化之后在进行后续的处理是有利的。离散化需要的是获得所有点的坐标，然后对所有的 x，y 坐标排序并去重，排序后得到的序号就是离散化后的坐标号。这里使用 STL 中的 map 来实现这一操作（map 内部实现是一个有序的红黑树）。X_coord 和 y_coord 变量分别记录了所有离散化后的坐标的原始坐标，discr_points 记录了所有点的离散化后的坐标。

Find_layout_L 和 find_layout_U 两个函数递归的求解所有的情况。从 root 的唯一的一个子节点开始，每次调用都在当前这个子树中，当前点和父节点之间的边选择 L（或 U）的情况下，求解最优的方案（方案即当前节点的子节点应该采用的选择）。每个点的每种选择的求解结果，放在 layout_l 和 layout_u 数组中，使用二进制编码，每一位代码某个子节点的选择（0 为 L，1 为 U）。

求解采用的方案是在地图上涂色。这时就用到了之前离散化后得到的整个地图。离散化后 hori_lines 和 verti_lines 分别代表了所有横向和纵向边，涂色也在这上面表示（用某条边上方或者右边的点的离散化坐标来表示该条边，所以这两个用 map 表示）。在对所有方案的涂色过程中，容易求出所有情况下 overlap 最大的做法。

对一个节点的所有子节点的所有可能情况点的穷举搜索使用 dfs 函数做递归。

2.2. ZRST

和 LRST 一样的需要做离散化以及建树。建树和离散化的过程和 LRST 相同。

然后对于每一个边，先预处理出其所有的可行的 Z 型 layout 备用。由于 Z 型的 layout 可以通过其拐点来唯一的确定其形态，于是就可以通过保存这个拐点坐标来表示一个 layout。

建好树之后，从树根开始，递归的调用（这里是 find_layout 函数）。对遍历的每个节点，先遍历其所有的子节点，然后对当前点枚举所有的可能的 layout（即当前节点的子节点的所有 layout 组合），这个用 dfs 函数实现。

3. 使用方法

使用命令行参数来控制程序的行为。“LRST”和“ZRST”分别运行两种算法。“test”用来生成新的测试数据。

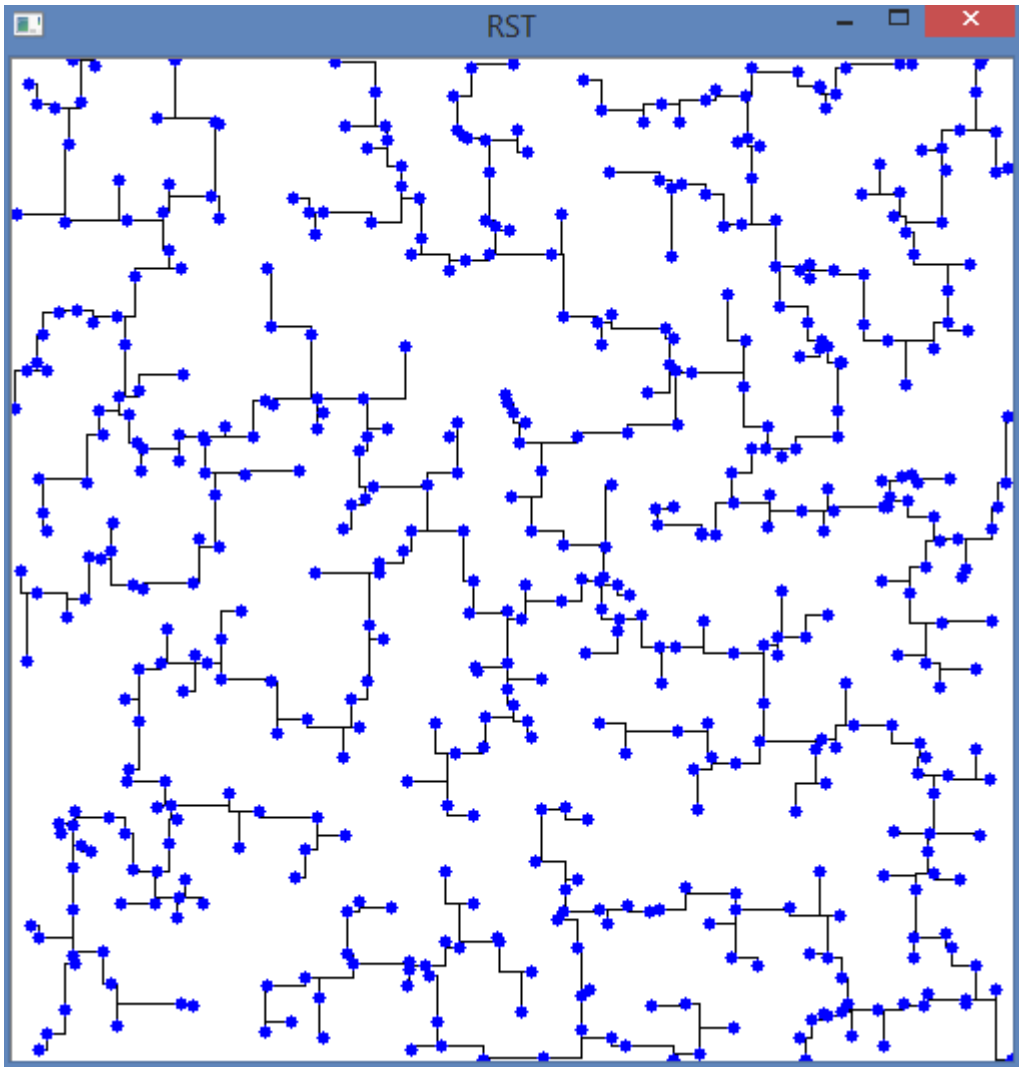
每次运行会显示求解的结果以及输出最后的总代价。

点的数目，文件名的常数可以在代码中修改（main.cpp 和 common.h）

4. 测试结果

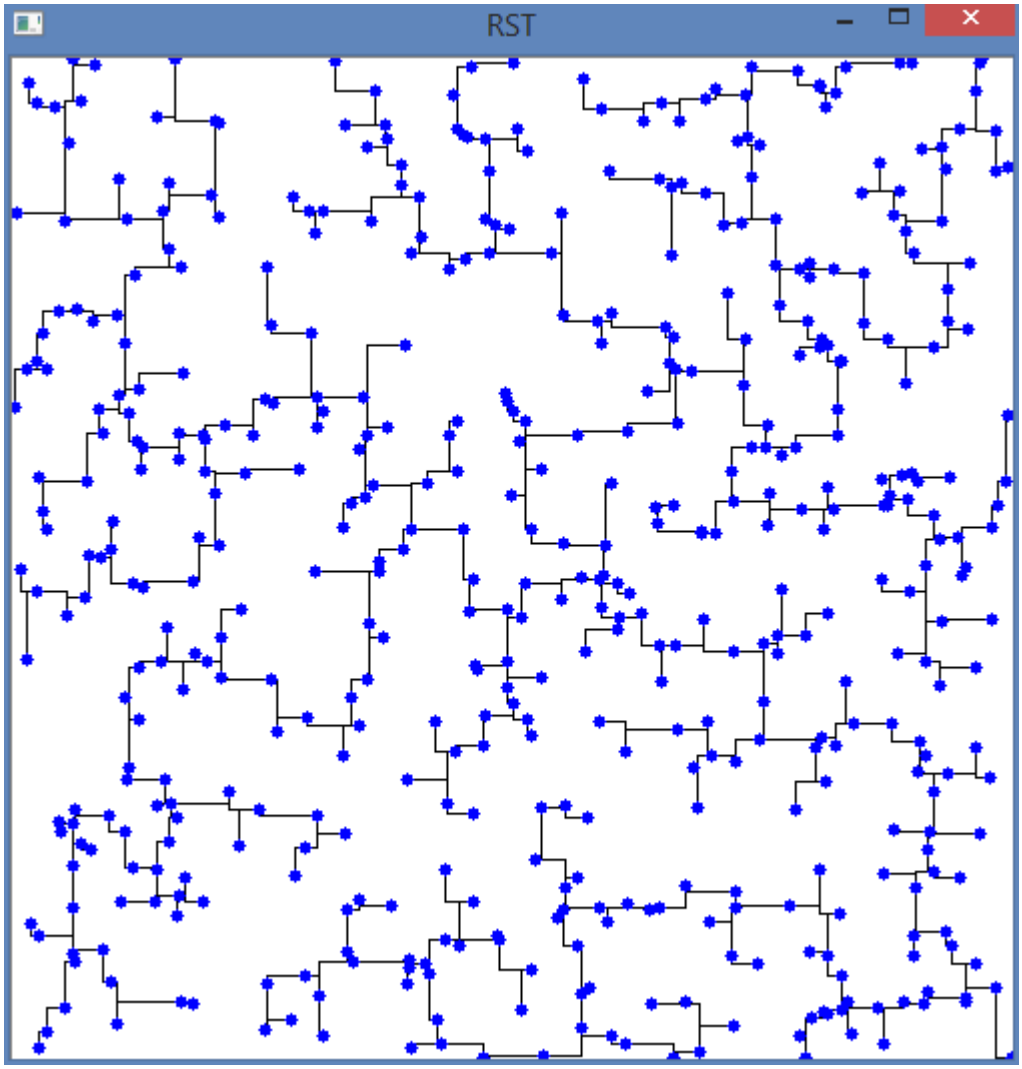
随机生成 500 个点运行，以下分别展示两种算法的运行结果。

4.1. LRST



总代价（即路径总长）：8480。

4.2. ZRST



总代价：8432

可以看出 ZRST 的计算结果比 LRST 要稍好一些。这是由于考虑了更多的情况。