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 庄天翼

邹昊

# 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。

# 技术实现

下面主要讨论LRST和ZRST的算法实现。

## 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。

## ZRST

和LRST一样的需要做离散化以及建树。

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。

# 使用方法

# 测试结果



