分组算法抽取文档

算法思路

输入输出

输入点集、障碍物、边界，输出分组结果，另有每组的最小最大数量等参数输入

分组思路

先不考虑障碍物，分组算法流程如下：

1. 对点集建立完全图 G
   图 G 的节点是点集中的点，每两个点之间的权值是两点的距离
2. 在完全图中生成最小生成树 MST
3. 将树的某条边打断变成两颗子树
4. 对子树重复 3 直到所有子树满足条件（例如包含点的数量不超过最大数量）

贪心思路

分组思路中第 3 步打断的操作，关键在于选择哪条边进行打断，本算法采用的是贪心算法，对每条边都有一个“评价值”，选取评价值最高的边进行打断。（注：每次打断后，子树要重新计算每条边的评价值）

1.4 分割

在 MST 中，每一条边若断开，都会将 MST 分为两个子树，再对子树不断分割，最终形成的森林中，每一棵子树包含的节点就被分为一组。采用**贪心策略**，分割一棵树时，先对树中的每一条边进行评估，若断开这条边形成子树 T1 和 T2，考虑：

- **位置关系：** T1 节点形成的最小外界矩阵 bbox1 中包含 T2 的节点数，越少越好；同理 bbox2 包含 T1 节点数越少越好

```
f1 = 3.0 / (2 + one_in_two + two_in_one);
```

- **数量均匀：** T1 与 T2 节点数的差的绝对值越小越好

```
f2 = 1.0 / (abs(size1 - size2) + 1);
```

- **组内凝聚力：** 使用树内所有边权的方差来体现凝聚力，T1 和 T2 的该值越小越好

```
f3 = 1000.0 / ((off1 + 1) * (off2 + 1));
```

- **被割边的权值：** 越大越好

```
f5 = 5.0 * tree[now].weight / (avg1 + avg2);
```

以一定的权重将各个评价值加起来得到评价函数，取评价最好的一条边进行分割，对子树进行迭代的分割操作，出口为节点数满足分组数量要求。

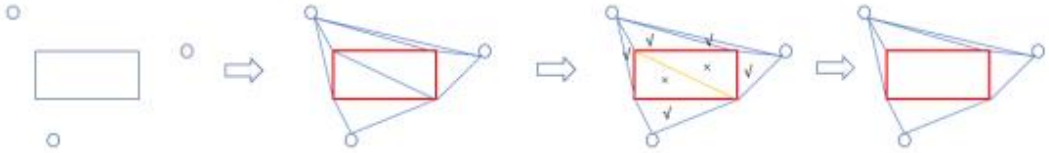
避障思路

以上是分组的思路，另外如果实际图纸需要考虑点位之间连线时障碍物的影响，则不能直接用点位之间的欧式距离来建图，引入一个新概念“避障距离”作为图中的权值，这个避障距离是一个对绕障碍连线距离的估算值

避障距离的计算：

使用**限制性 Denaulay 三角化**（Constraint Denaulay Triangulation, **CDT**）

1) 将设备点位插入 CDT
2) 将防火分区和障碍物的边，作为**限制边**插入 CDT，限制边一定会出现在 CDT 中
3) **标记非障碍面片：**以设备点位所在的三角面片为起点，向三边邻接的面片拓展并标记为非障碍，若为限制边则不向那一边拓展。拓展结束后，在障碍物内部或防火分区外部的三角面片未被标记，即为障碍面片
4) **删去越障边：**遍历 CDT 中所有边，若边所关联的两个三角面片都为障碍面片，则该边为越障边，删去，否则保留。



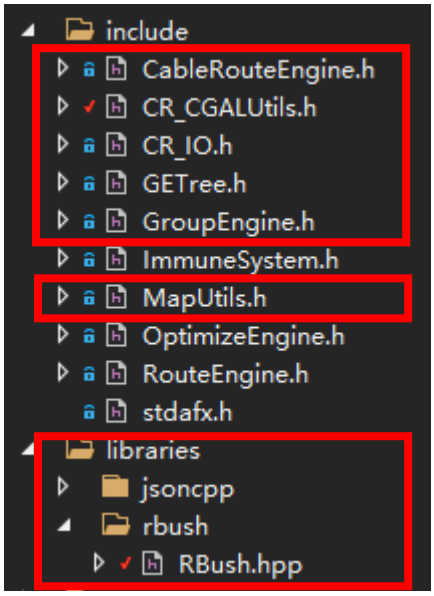
a) 点位与多边形障碍物　b) 生成CDT，红色边为障碍物形成的限制边　c) 标记面片，黄色边两边都没被标记，为越障边，删去

1.2 建立完全图

在 CDT 的顶点中，有设备点，也有障碍点（障碍物和防火分区轮廓顶点）。CDT 可以视为一个图 G_cdt，要从该图中抽取一个**子图 G**，G 是只包含设备点的完全图，包含每个设备之间的距离信息

1) **添加直接可达边：**CDT 中有些点之间没有直接相连，但其实他们连接起来并没有穿过障碍物，将这种边加入 G_cdt 中（包括设备点和障碍点）
2) **加权：**对 G_cdt 中的每条边，根据业务逻辑，比如穿过中心线或穿过房间框线时，进行加权，表示不希望走这条边
3) **抽取子图 G：**在 G_cdt 中，计算所有设备点之间的最短路径。此处采用的方法为，以每个设备点为起点做一次 Dijkstra 算法，取结果中设备点到起点的最短路径。

相关代码



先看 CableRouteEngine.cpp，这是之前项目所用的接口。
routing 函数的截图部分是调用分组的相关代码

```
string CableRouter::CableRouteEngine::routing(string datastr, int loop_max_count, int iteration_count)
{
    if (loop_max_count < 1)
    {
        return "error: loop max count < 1";
    }
    if (iteration_count < 1)
    {
        return "error: iteration count < 1";
    }

    MapInfo map;
    GroupParam param;
    GroupEngine ge;

    // parse geojson file
    map = read_from_geojson_string(datastr);
    preprocess(map);
    if (map.devices.size() == 0)
    {
        return "error: no Valid Wiring Position";
    }
    else if (map.powers.size() == 0)
    {
        return "error: no Valid Power Position";
    }
    else if (map.area.info.boundary.size() == 0)
    {
        return "error: no Fire Apart";
    }

    // grouping
    param.max_dev_size = loop_max_count;
    param.min_dev_size = max(1, loop_max_count / 2);
    param.weight_pos = ALPHA;
    param.weight_even = BETA;
    param.weight_cohesion = OMEGA;
    param.weight_big = DELTA;
    param.weight_cut_len = GAMMA;

    vector<vector<int>> group_info = ge.grouping(&map, &param);
```

首先读取输入的 geojson 转换成自定义的 MapInfo 数据结构，然后初始化一个 GroupParam 作为参数合集，最后调用 GroupEngine 类的方法 grouping 得到分组结果。
返回值为 vector<vector<int>>，这里的 int 是点位在 MapInfo 中对应的 Id

其余文件的内容：
jsoncpp:：json 文件的读取
rbush：空间搜索库
CR_CGALUtils：CGAL、rbush 相关操作打包
CR_IO：json 文件读取后转换成自定义的数据结构 Mapinfo
MapUtils：定义 Mapinfo 以及相关操作
GroupEngine：分组整个流程的相关代码
GETree：定义分组所用的树的数据结构和相关操作

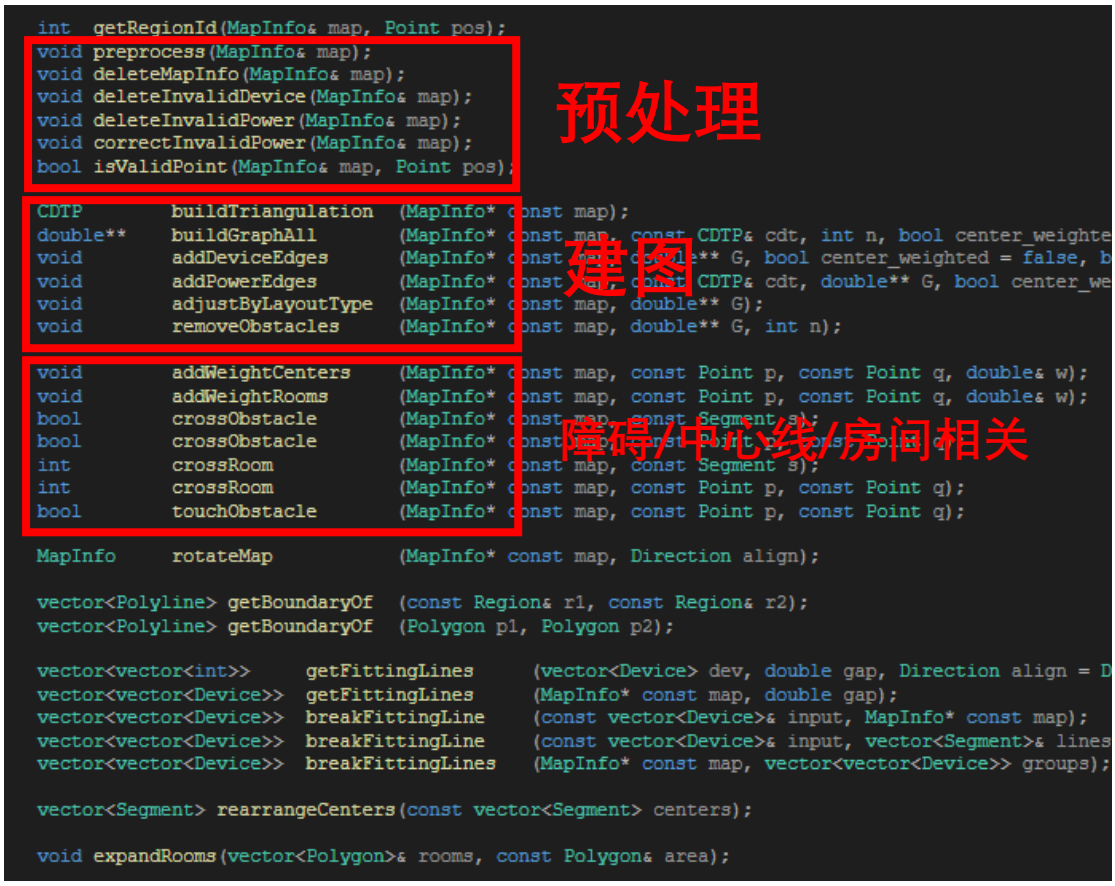
抽取思路

如果业务逻辑比较相似，MapInfo 的数据结构是可以复用的话，抽取相对简单，只需要将对应的输入转成 MapInfo（参考 CR_IO），然后调用相关函数即可，或者重新定义自己的 MapInfo，将不相关的操作都删除，关键在 MapUtils 文件。目前的数据结构如下：

```
struct MapInfo
{
    vector<Power> powers;
    vector<Device> devices;
    vector<Segment> centers;
    vector<Polygon> holes;
    vector<Polygon> rooms;
    vector<Region> regions;
    vector<Segment> borders;
    FireArea area;
    SegBush* cen_line_tree;
    PEBush* hole_tree;
    PEBush* room_tree;
};
```

1) 仅考虑分组的话，不需要 powers
2) devices 是要分组的点集，ID 与在 vector 中的索引保持一致
3) centers 和 rooms 视业务需求而定，这里代表中心线和房间框线，连线穿越这些线时会增加距离权值
4) holes 是图上的障碍物，连线不能穿越的洞口、剪力墙等
5) regions 和 borders 是连线方向相关的，应该是不需要的
6) area 是整个图纸的边框
7) 3 个 bush tree 是对应的空间搜索树，在 MapInfo 建立时随之建立

相关函数：



没框起来的是与分组无关的。

1) 如果希望更加通用，可以定义一个新的更加通用的 MapInfo 数据结构，然后根据新的数据结构将不需要的函数删去，对函数内部的相关代码进行改写

2) 如果是简单使用看效果，可以将上述 MapInfo 不需要的内容设置为空（vector 和 rbush 只初始化，不加内容），应该也是能跑通的

3) 贪心策略的评价值计算在 GroupEngine 的 evaluate 函数中，可以进行参数调整或者新增想要的评价因子