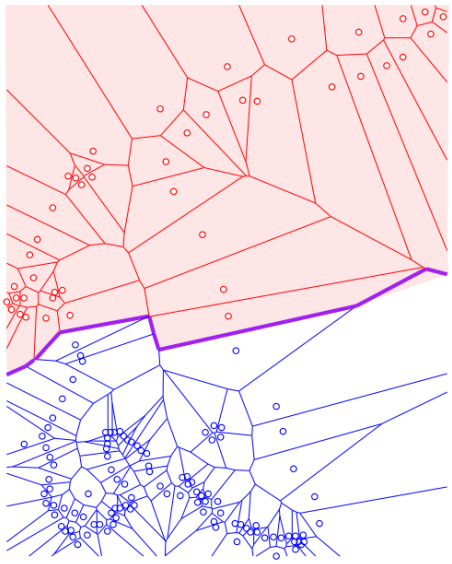
Report-Voronoi Separator

1. 问题描述

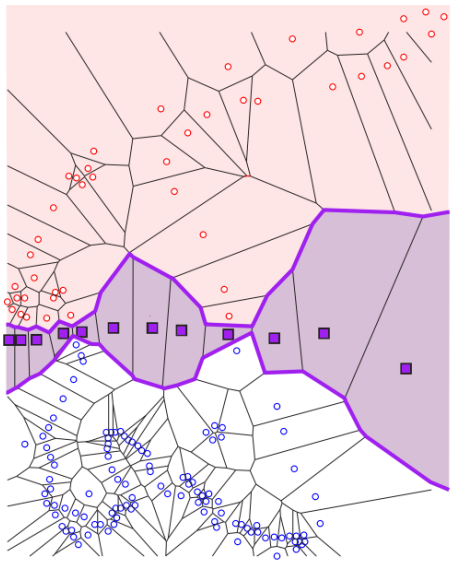
问题一（Compute a Balanced Voronoi Separator）：给出空间中的若干个点，计算一个分割集（Separator），该集合中的点满足：将该分割集插入空间中后，空间中原来的点的被这个分割集中的点分为数量相近的两部分，两部分点的 Voronoi 图不相交，两个部分之间点所在的 Voronoi Cell 不存在公共边界。

问题二（Polynomial Time approximation Scheme(PTS) Separator）：给出空间中的若干个点，这些点分别属于两个集合。需要在多项式时间内计算一个分割集，该集合中的点满足：将该分割集插入空间中后，在包含新加入的分割集中的点的 Voronoi 图中，分属原来两个不同集合的点 Voronoi 图不相交，每个点所在的 Voronoi Cell 不存在公共边界。

(A)



(B)



图例：(A) 红色的 Cell 和白色的 Cell 分属两个不同集合，(B)插入 Separator 之后，Separator 的点以紫色正方形表示，Separator 所在的 Cell 为紫色 Cell.

2. 算法说明

✧ 问题一：计算一个 Balanced Voronoi Separator

定义 1:对于球b，它的**覆盖数**定义为能够覆盖自身b，且半径为自身一半大小的球的最小数量。度量空间的**倍增常数**定义为覆盖所有可能的球的最大数，记为c_{dbl}^{d}。
定义 2:集合$Y \in X \in \mathbb{R}^d$在集合X中是$\ell - dense$的， 如果对于任意$p \in X$，存在$s \in Y$，使得$\| p - s \| \leq \ell$
引理 1:给定一个任意球$\mathbb{S}$以及一点$p \in \mathbb{R}^d$且$p \notin \mathbb{S}$,那么可以在常数时间内计算得到一个点集$Q \subseteq \mathbb{S}$，使得 Voronoi Cell 集$\mathcal{C}_{Q \cup p}(p)$与$\mathbb{S}$不相交，且$|Q| = O(1)$,同时将$Q$记为$blockerSet(p, \mathbb{S})$

问题描述：给定d维空间$\mathbb{R}^d$中由n个点组成的集合P，计算一个平衡的 Voronoi 分割集，该集合的大小为$O(n^{1-1/d})$
算法流程:
输入为d维空间$\mathbb{R}^d$中n个点的集合P，算法执行如下:
a) 令$c_d = c_{dbl}^{d}+1$，令球$b(\psi, r)$表示包含P中n/c_d个点的最小球，其中$\psi \in \mathbb{R}^d$
b) 从区间$[r, 2r]$随机选择一个半径r'，令$b'(\psi, r')$
c) 令$P_1 = P \cap b'$且$P_2 = P \setminus b'$
d) 令$\ell = r'/n^{1/d}$，在球$\mathbb{S} = \partial b'$上计算一个大小为$O(n^{1-1/d})$的$\ell - dense$集合Z
e) 对于每一个$p \in P$且与$\mathbb{S}$距离小于ℓ的点，将$blockerSet(p, \mathbb{S})$插入集合$Z$
以上步骤之后最终的集合$\boldsymbol{Z}$即为所求的 Voronoi 图分割集
关键技术点:
1) 对于步骤 d) 如何计算集合Z，理论上可以在$O(z)$的时间内完成，其中$z = |Z|$
2) 对于步骤 e) 在将$blockerSet(p, \mathbb{S})$插入集合$Z$时，如何维护$Z$中元素的有序性

结论:
令P为d维空间$\mathbb{R}^d$中n个点的集合,则可以在线性时间内,计算得到一个球$\mathbb{S}$以及$Z \subseteq \mathbb{S}$,满足:
1) $|Z| = O(n^{1-1/d})$
2) $\mathbb{S}$中至少包含cn个P中的点
3）$\mathbb{S}$外至少有cn个P中的点
4）Z是一个将位于$\mathbb{S}$内外的属于P中的点划分开来的分割集
此处，$c > 0$是一个仅取决于维度d的常数

✧ 问题二：多项式时间计算一个最优 Voronoi 分割集的近似算法

定理 5.2：对于一个$\mathbb{R}^d$中的点集 X，并且令$k > 0$是一个足够小于|X|的数，可以在 O(|X|)的时间内计算出一个 O($k^{1-1/d}$) 大小的集合Z和一个内部包含$\theta(k)$个 X 中的点的球面

定理 4.10：对于$\mathbb{R}^d$中的 n 个点组成的点集 P，对于 P 中不相交的一个分割(P_1, P_2)以及一个参数 b，满足存在 P 的 Voronoi 分割至多只有 b 个点，则最小 Voronoi 分割可以在$n^{O(b)}$的时间内计算出来

记号：

其中 feature、sites 等解释参照文献中的定义

$$\mathcal{F}_{bad}(P, P_1, P_2) = \{f \in features(\mathcal{V}(P)|sites(f) \cap P_1 \neq \emptyset \; and \; sites(f) \cap P_2 \neq \emptyset\}$$

$$\text{pencil(f)} = \{\text{ball}(\mathbf{p}, \text{d}(\mathbf{p}, \text{sites(f)}))|\text{p} \in \text{f}\}$$

$$\mathcal{B}(P, P_1, P_2) = \cup_{\text{f} \in \mathcal{F}_{bad}(P, P_1, P_2)} \text{pencil(f)}$$

问题描述

给定一个$\mathbb{R}^d$中的点集合 P，以及 P 的一个分割(P_1, P_2)，需要在$n^{O(d)}$次方（多项式）时间内计算出一个 Voronoi 分割集 (Separator) Z, Z 的大小是取得该分割的最小 Voronoi Separator 集的常数倍

算法流程

首先需要构造一个贪心算法 (GreedySeparator)， 先计算出集合(P_1, P_2) 的一个 Separator，从一个空集X开始构造，每次计算出$P \cup X$的一个 Voronoi 图，然后计算出$\mathcal{F}_{bad}(P \cup X, P_1, P_2)$，对于每个$f \in \mathcal{F}_{bad}(P \cup X, P_1, P_2)$计算出 pencil(f)中的极小球。令$b = ball(\boldsymbol{p}, r)$为其中出现的最小的球。令$P_b$为常数大小的点集，且任何与 b 有相交的球都会与$P_b$相交。$P_b$可以通过将一个 r-dense 的点集散布在扩张的球$ball(\boldsymbol{p}, 2r)$中得到（即每个$ball(\boldsymbol{p}, 2r)$中的点到P_b内至少有一个点的距离不超过 r）。此时P_b总是具有常数大小，此时将P_b加入X中，循环往复，直至计算出一个 Separator。

然后执行一下算法流程来计算出一个更好的 Separator：

1) 首先使用 GreedySeparator 计算出一个 Voronoi Separator X，令$k = O(\frac{1}{\epsilon^d})$
2) $i \coloneqq 1$
3) $X_1 \coloneqq X, \mathcal{Z}_1 \coloneqq \emptyset$
4) 当$X_i \neq \emptyset$时，重复以下流程
   a) 利用定理 5.2 计算出一个球面$\mathbb{S}_i$以及$\mathcal{Z}_i^{\mathbb{S}}$
   b) 令b_i为$\mathbb{S}_i$所对应的闭球
   c) $Q_i \coloneqq (P \cap b_i) \cup Z_i \cup Z_i^{\mathbb{S}}$
   d) $P_{i,1} \coloneqq b_i \cap P_1$
   e) $P_{i,2} \coloneqq b_i \cap P_2$
   f) $\Xi \coloneqq b_i \cap X_i$
   g) $Z_i' \coloneqq$ 利用定理 4.10 计算出的对于 $\mathcal{B}(Q_i, P_{i,1}, P_{i,2})$上的最小碰撞集
   h) $X_{i+1} \coloneqq X_i \backslash b_i$
   i) $Z_{i+1} \coloneqq Z_i \cup Z_i^{\mathbb{S}} \cup Z_i'$
   j) $i \coloneqq i + 1$
5) 输出一个 Separator $\mathcal{Z}_i$

关键技术点：

1) 对于 Separator 的贪心构造法，和之前的 Separator 构造算法一样，每次都会更新 Voronoi 图，实际上每次迭代需要在上一次生成的 Voronoi 图上加入此次的 Separator 中的新点，需要维护一个适合更新和加入新点的数据结构。另外计算 pencil 中的最小球时由于要计算$\mathcal{F}_{bad}$所以还需要对 Voronoi 求出各个维度下的 feature，过程较长。

2) 定理 5.2 需要确定一个足够小的 k 值，所以在实际具体计算的时候需要根据实际的数据情况来确定 k 的大小，也许需要对 k 和|X|的常熟关系进行分析

3) 定理 4.10 实际上给出了一个具体的复杂度上界的结论，但是在证明中依然没有给出完整清晰的算法流程，所以在使用这条定理时仍然需要理解如何通过合适的方法计算碰撞集来达到预期的复杂度上界。

3. 数据结构与模块设计

- 最小 k 包围圆[1]

问题描述: 给定平面上n个点的集合S，找出一个半径最小的圆(ϕ, r)，使得其至少包含S中的k $(k \leq r)$个点。

我们首先介绍一些术语。给定平面中的无重点集S。对于平面中任意一点p和半径$r > 0$，定义以p为圆心，半径为r的圆为$C_r(p)$，对应的封闭圆面为B_r(p)。对于任意的$r > 0$，记圆排列C_r(p)，$p \in S$为$\mathcal{A}(S, r)$。给定一个圆排列$\mathcal{A}$和一个点x，记x在$\mathcal{A}$中的深度为$depth_{\mathcal{A}}(x)$，表示$\mathcal{A}$中包含点x的圆的数量。因此我们也定义$depth(\mathcal{A}) = max\{depth_{\mathcal{A}}(x) \mid x \in \mathbb{R}^2\}$。 因而有如下的两个定义:

定义 1:给定一个点集S，点x和正数r，$|S \cap \mathrm{B}_r(x)| = depth_{\mathcal{A}(S,r)}(x)$

定义 2:给定一个点集S，点x和正数r，$max\{|S \cap \mathrm{B}_r(x)| : x \in \mathbb{R}^2\} = depth(\mathcal{A}(S, r))$

基于以上的定义，最小 k 包围圆问题实际可以等价转化为，寻找一个常数r使得$depth\big(\mathcal{A}(S, r)\big) = k$。对于每一个$r$，我们需要对给定点集$S$中的每一个$C_r(p_i)$计算包含在圆内的点的最大深度，所有的输出深度的最大值即为$depth(\mathcal{A}(S, r))$，算法流程如$Calc - Depth(r, i)$

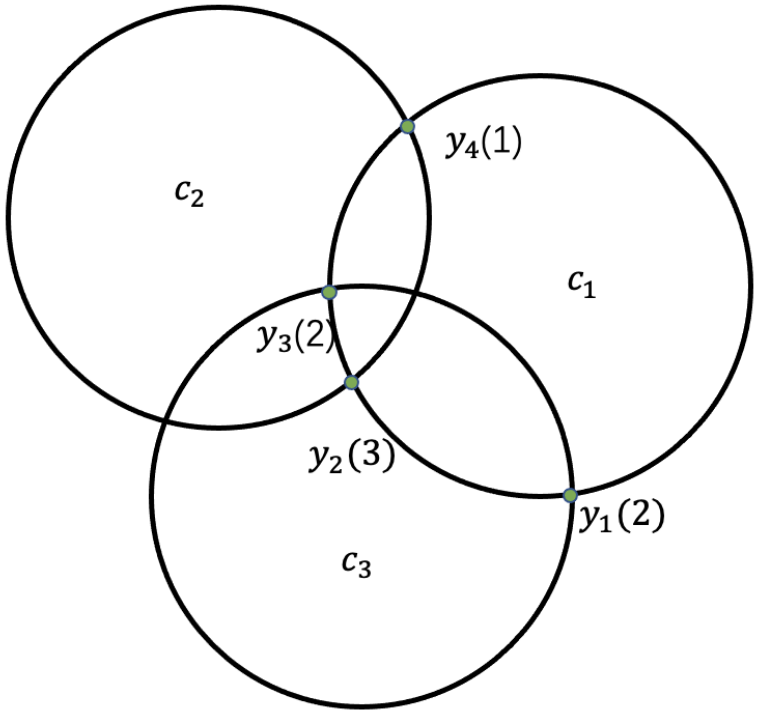
$Calc - Depth(r, i)$**算法流程**:

f) 找出与$C_r(p_i)$相交的所有圆$C_r\big(p_j\big)$。如果找不到，返回 1，算法结束。

g) 计算$C_r(p_i)$的最左点的深度

h) 将$C_r(p_i)$与所有$C_r\big(p_j\big)$的交点按照顺时针方向排序，起始点从上一步骤的最左点开始

i) 扫描每一个交点，计算其各自的深度。需要注意的是，当我们扫描到交点ip时（假设该点是圆i和圆m的交点），如果顺时针通过该点进入圆m，则深度+1，反之-1。 详细说明如下图：圆c_2和c_3与圆c_1一共相交于 4 点，顺时针排序排序分别为y_1，y_2，y_3，y_4，其深度分别为 2，3，2，1



j) 返回以上步骤中计算的最大深度

在计算最小 k 包围圆的过程中需要反复调用以上$Calc-Depth(r,i)$算法，从而确定最优最小半径r^*。为快速求解r^*，我们采用二分搜索的策略，即把圆i相关的$n-1$个距离对排序，通过每次选定r之后求解的最大深度与 k 的大小关系，快速确定下一个 r 的搜索范围

算法设计：

数据结构：

a) point.h: 定义了基本的点 struct Point，其中存储了点的坐标 x 和 y，为减少计算过程中的精度损失以及增强系统的鲁棒性，采用 long double 数据类型。此外定义了一些基本的数据操作，诸如向量叉积，向量的模，单位化以及向量的加减乘除等各种运算

b) intersection.h: 定义了两圆交点的基本数据结构，其中存储了交点自身的坐标，以及所在的圆对应的 id

c) circle.h: 定义了基本的圆 struct Circle，其中存储了圆心 center，与其他圆的交点 ips，与其他圆心的距离 dists 以及自身 id（便于后续交点 ips 的深度），此外还定义了求解与其它圆交点的 getIntersection 函数

d) k-enclosing.cpp: 定义了主函数 kEnclosing，按照上述流程的设计反复调用$Calc-Depth(r,i)$函数求解最优半径r^*，并且返回对应的圆心。此外一些预处理操作例如极角排序以及 utils 函数也定义在该文件中。

- Voronoi Diagram

1 - 构思

PA23 已用随机增量法实现了 Delauny Triangulation (DT),取其对偶图即可获得 Voronoi Diagram (VD)。实现 Delauny 三角剖分的数据结构是 DCEL,这里只给出 DECL 数据结构而不再赘述 Delauny 三角剖分的生成，重点介绍对偶图(VD)的生成。

DCEL

```
Vertex {
    id
    x, y
    inc // incident to edge
    faces // bucket
}
Face {
    id
    inc // incident to edge
}
HalfEdge {
    id
    inc // incident to face
    ori // incident to original vertex
    succ
    pred
    twin
    vertices // bucket
}
```

完全从 DT 转到 DCEL 表示的 VD 是一件困难的事情, 所幸我们的 Separator 算法并不需要完整的 VD 信息，先给出取 VD 数据结构。

```
Vertex/Site {
    x, y
}
Voronoi {
    Site    // center of the cell
    Vertex:    // endpoints of the bisector (0-feature)
    gv:    // DAG of vertexes
    gs:    // DAG of sites
    mp_site_vertex:    // a map from site to vertexes
}
```

---

2 - 原理

DT-VD 有以下几个对偶:

* vertex - site
* Circumscribed circle center (CCC) - vertex

因此我们可以遍历每一个面 f,将 f 上每一个点 v 存为对偶图的 site,然后计算 f 外接圆圆心 ccc 并存为对偶图的 vertex。根据 DT 中 vertex 的邻接关系生成 gs，face 邻接关系生成 gv。至于 mp_site_vertex，可以在遍历面的时候建立 site 到 vertex 映射。

3 - 复杂度估计

3.1 时间复杂度

DT 时间复杂度为 O(nlogn)，转化对偶图需要遍历每一个面 O(F)，合计 O(nlogn + F)，其中 F 为 DT 中三角形的数目，总期望不超过 O(9n+1)，因此总复杂度为 O(nlogn)。

3.2 空间复杂度

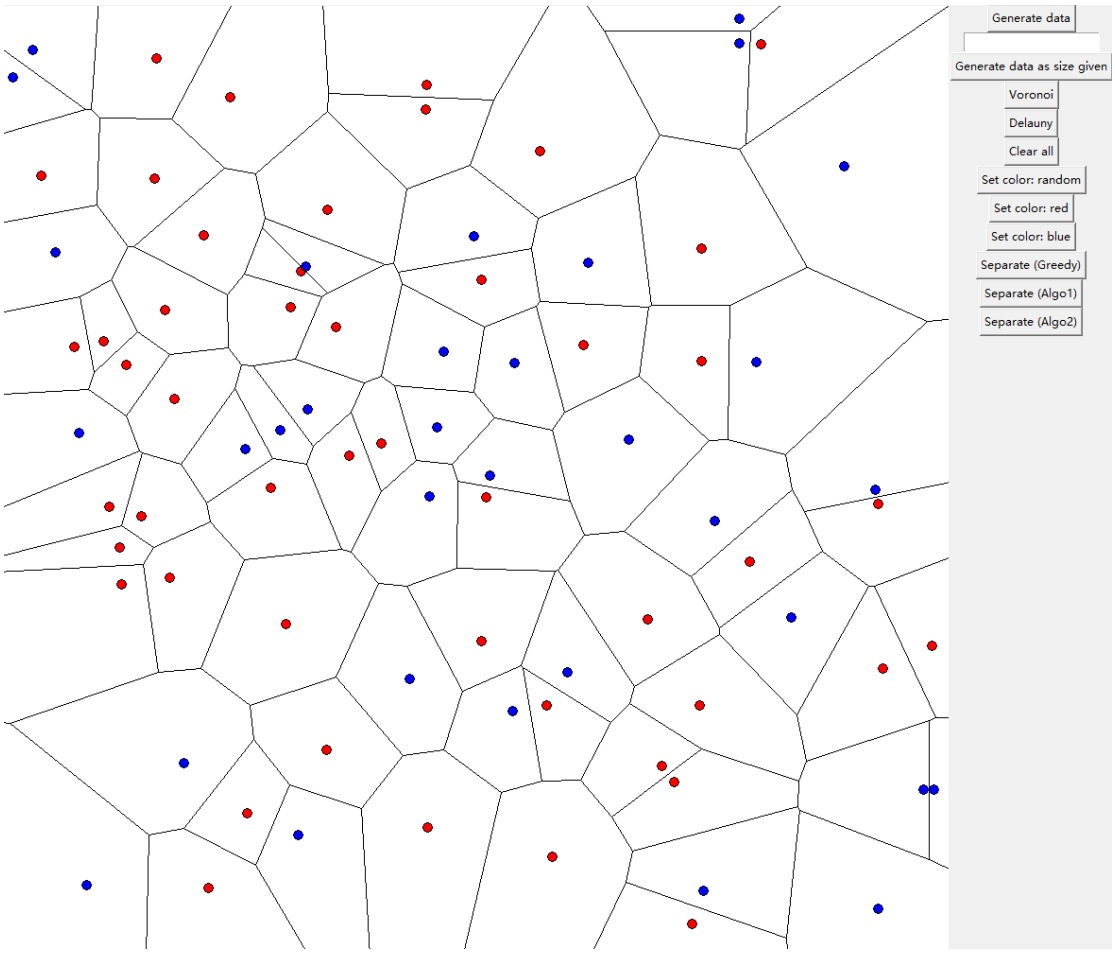
与 DT 一致为 O(n)。

- Graphical User Interface(GUI)

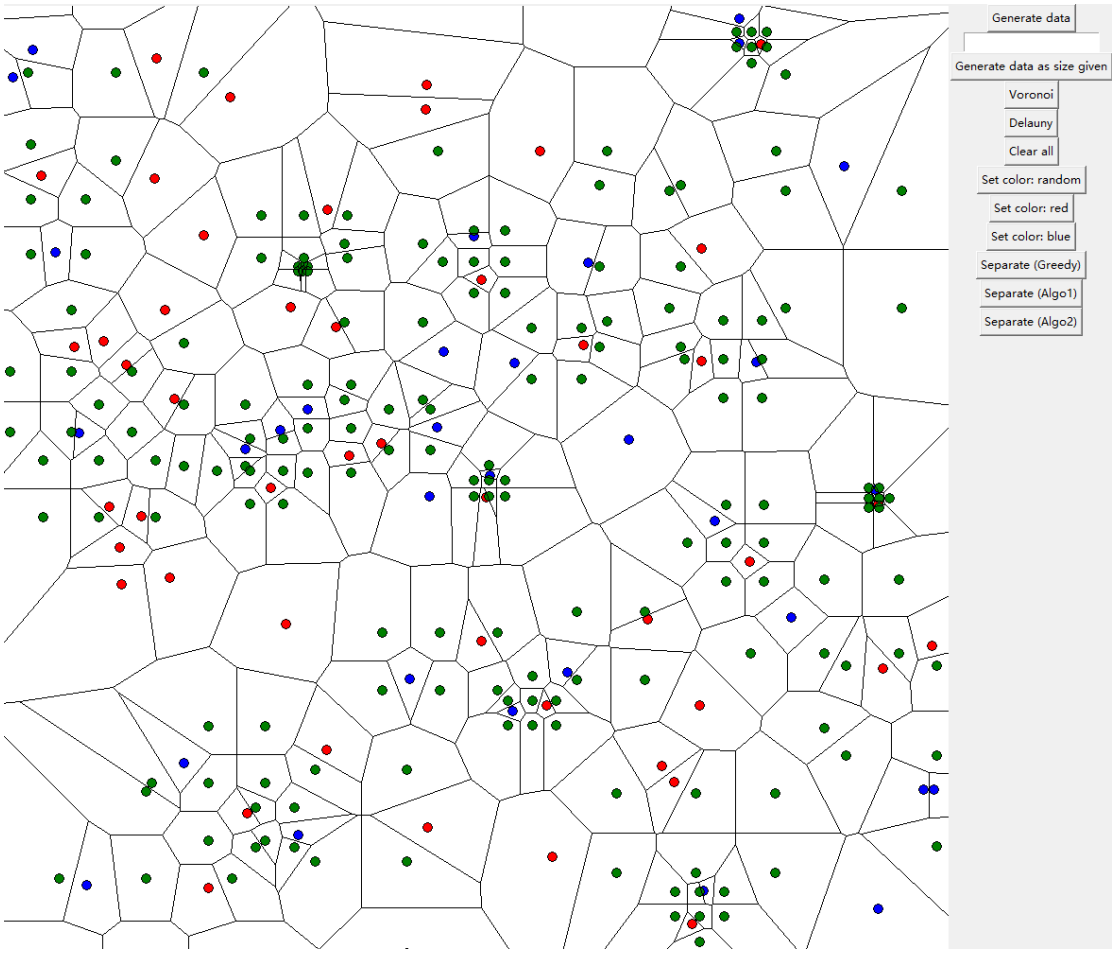
GUI 使用 tkinter 实现, tkinter 简单易用，其 canvas 部件能在画布上绘制点、线、面。为按钮设置好回调函数后可以调用各模块，将输出的数据以一定方式绘制在画布上。

主程序通过创建进程调用各个模块，用文件进行通讯。模块分为两大类。
* Voronoi Diagram 生成
* Separator 生成点(guards)分割 Voronoi Diagram

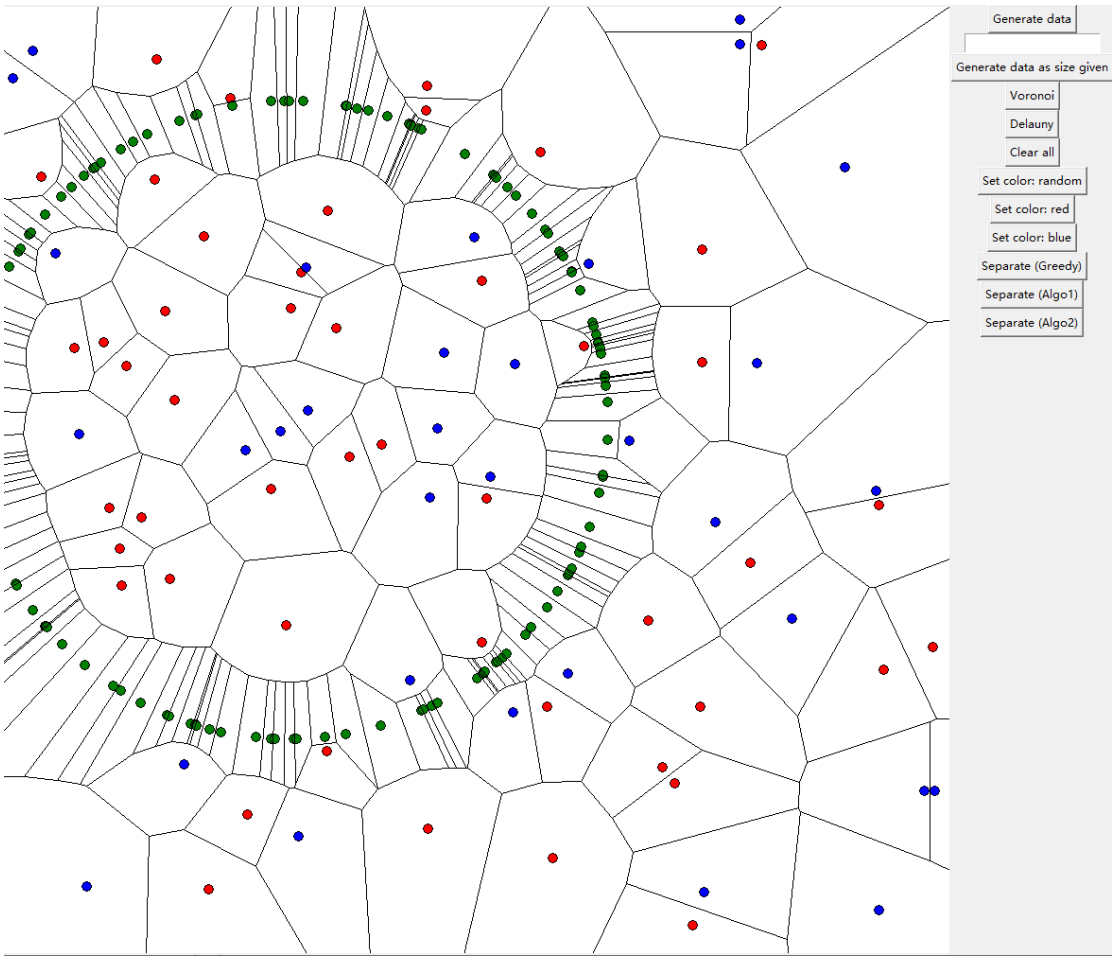
主程序根据用户发送的指令控制模块通讯、算法启动、绘制画面。



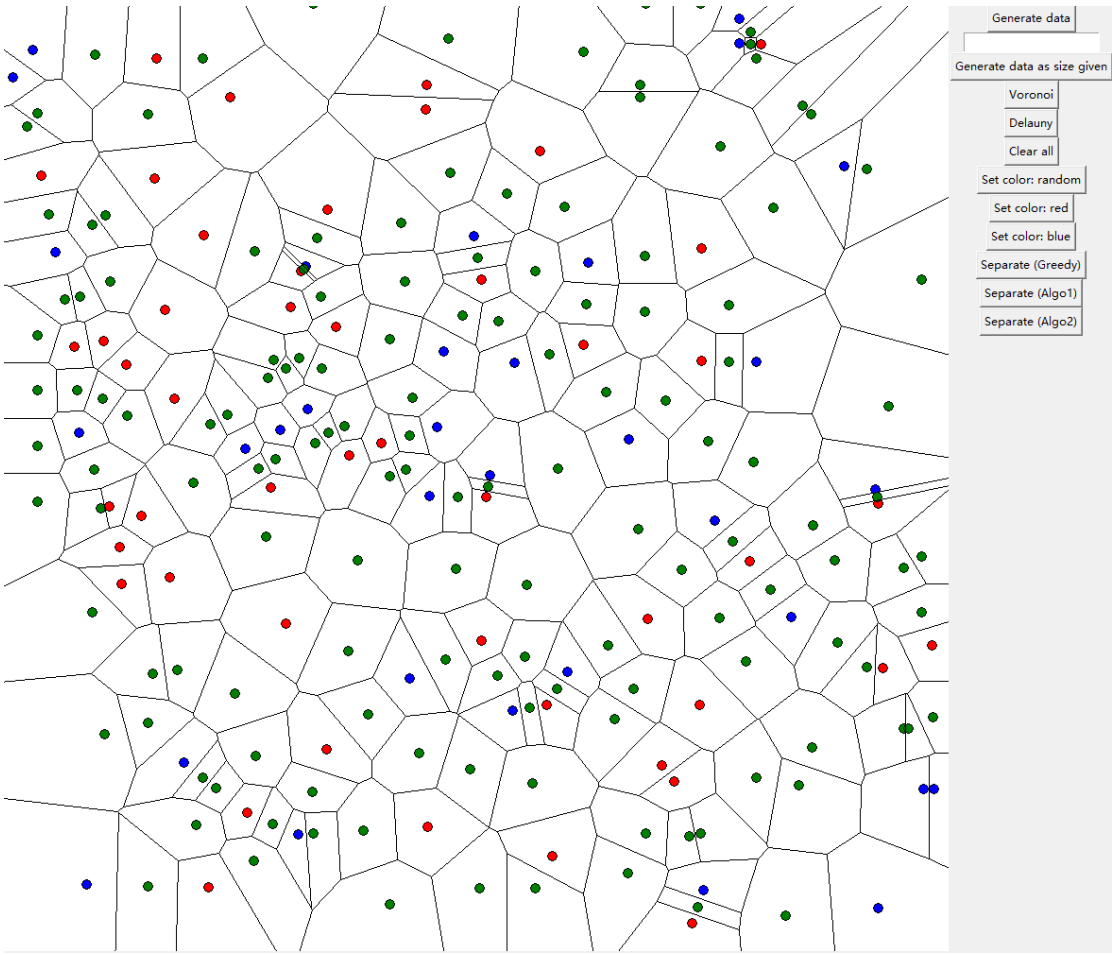
3.1 GUI-Generate data - Voronoi



3.2 GUI – Separate - Greedy



3.3 GUI-Separate-Algo1



3.4 GUI-Separate-Algo2

- Input & Output

Voronoi 图的输入：输入 n 点及坐标，返回绘制 Voronoi Diagram 需要的信息

n
x_1 y_1
...
x_i y_i
...
x_n y_n
z_1 ... z_i ... z_n

n: 输入点个数
x_i y_i: 第 i 个点坐标
z_i: 第 i 个点分类, z_i = {0, 1}

Voronoi 图的输出

n
(x_1, y_1)
...
(x_i, y_i)
...
(x_n, y_n)
(v_1.x, v_1.y) (u_1.x, u_1.y) ... (u_k.x, u_k.y)
...
(v_i.x, v_i.y) (u_1.x, u_1.y) ... (u_k.x, u_k.y)
...
(v_m.x, v_m.y) (u_1.x, u_1.y) ... (u_k.x, u_k.y)

n: site 个数
(x_i,y_i): 第 i 个 site 坐标
(v_i.x,v_i.y): 第 i 个 vertex 坐标
(u_k.x,u_k.y): 与(v_i.x,v_i.y)相互连接的第 k 个 vertex 坐标

Separator（Balanced Separator 和 PTSSeparator）的输入：输入 n 个点及坐标，输出 guards 数目及坐标。Separator 分为三类：Greedy/算法 1/算法 2，它们的输入输出接口是一致的

n
x_1 y_1
...

x_i y_i

...

x_n y_n

z_1 ... z_i ... z_n

n: 输入点个数

x_i y_i: 第 i 个点坐标

z_i: 第 i 个点分类, z_i = {0, 1}

Separator 的输出

---

n+m

x_1 y_1

...

x_i y_i

...

x_n y_n

x_{n+1} y_{n+1}

...

X_j y_j

...

x_{n+m} y_{n+m}

z_1 ... z_i ... z_{n+m}

n: site 个数

m: guards 个数

(x_i,y_i): 第 i 个 site/guard 坐标

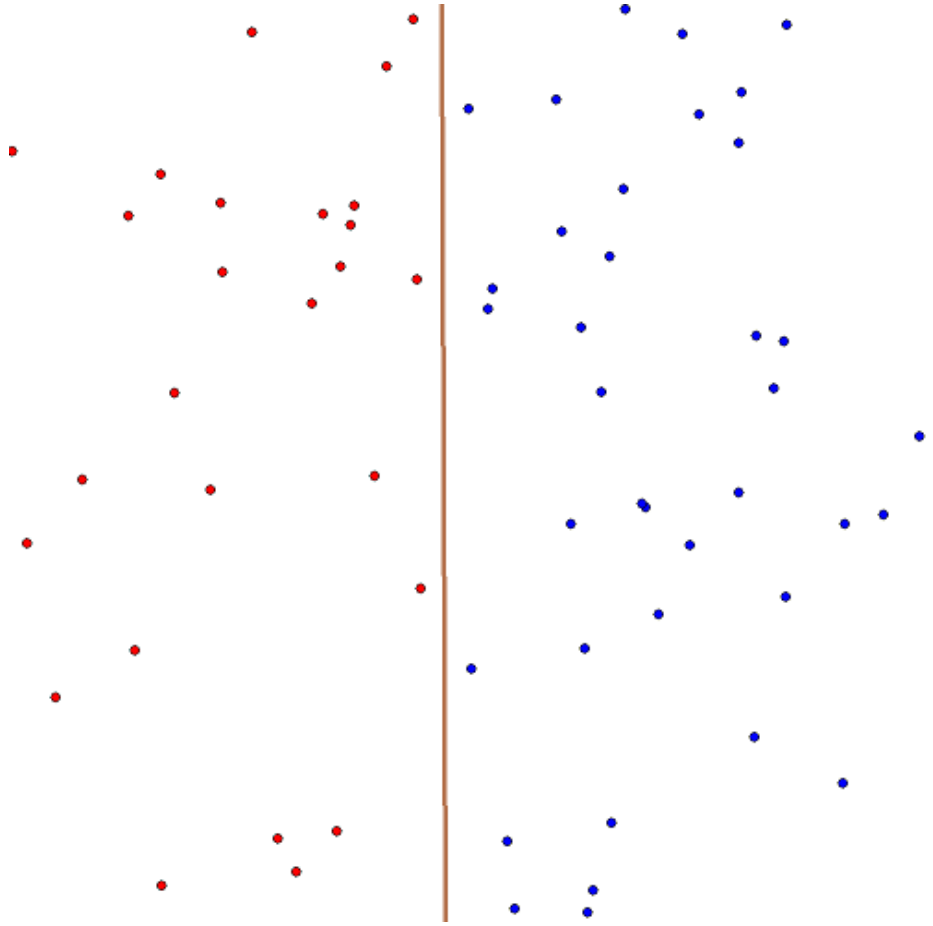
z_i: 第 i 个 site/guard 分类 z_{site} = {0, 1}, z_{guard} = {2}

---

4. 算法性能测试

算法的测试数据分为三种分布的测试数据 Scheme1，Scheme2 和 Scheme3，Basic 测试数据可以使用 GUI 生成，测试数据在 project/testcase/in 中，其中 (1+ϵ)-Approximation PTS Separator 中ϵ选用值为 0.1。在以下测试中，每个同类型的数据会有十组数据，运行结果以表格的方式展示，表格的第一行为生成的点数，第一列为使用的算法，表格中的数据为总运行时间（单位:s）。例如，第二行第三列的数据表示 Balanced Separator 在 500 个点的数据下 10 组数据的总运行时间为 84.04 秒。

- 对于 Scheme1 数据，数据生成方式为生成一条斜率为 k 的直线，并且在平面上随机生成 **n** 个点，其中落在直线划分出的不同半平面的点分属不同集合



图例:scheme1 类型的数据点

算法\n	200	500	1000	2000	5000
Balanced	6.35	84.04	656.56	5195.23	-
Greedy	1.59	5.31	16.74	78.24	639.65
PTS	2.13	7.67	21.674	90.72	711.98

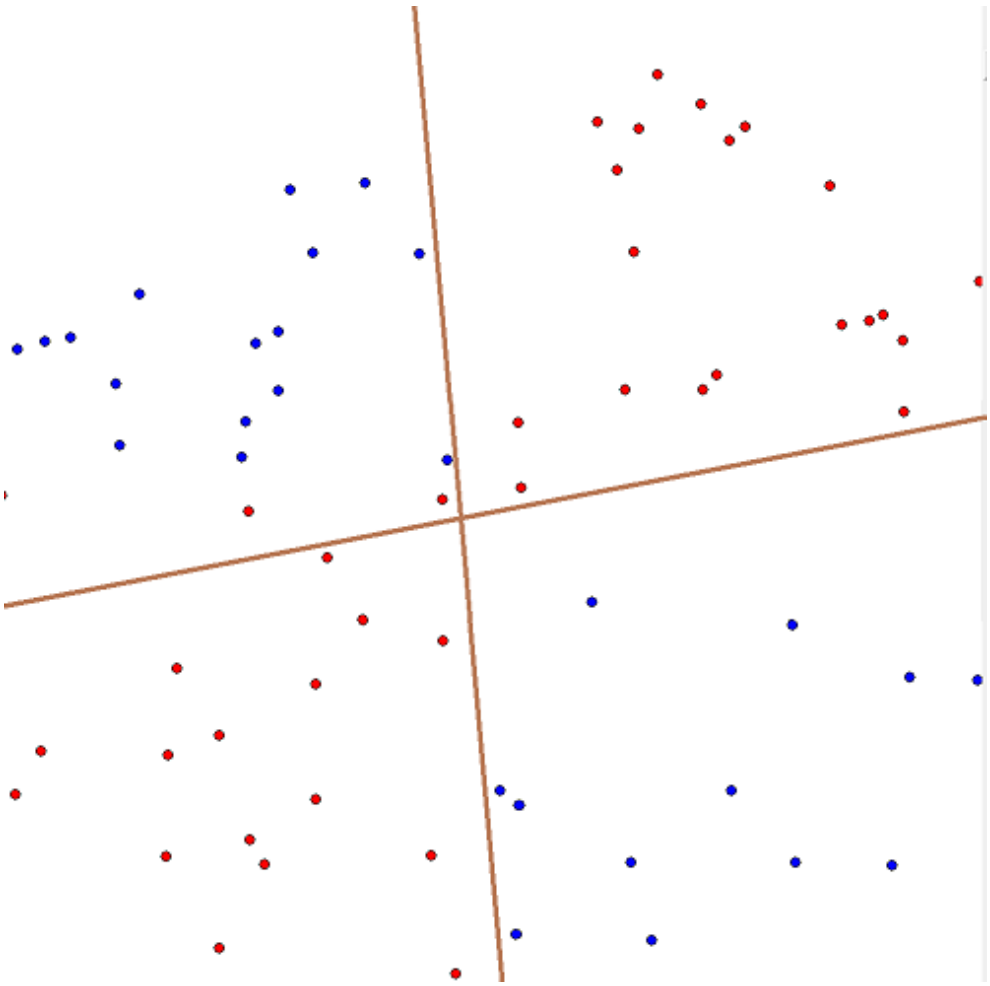
-

-

-

-

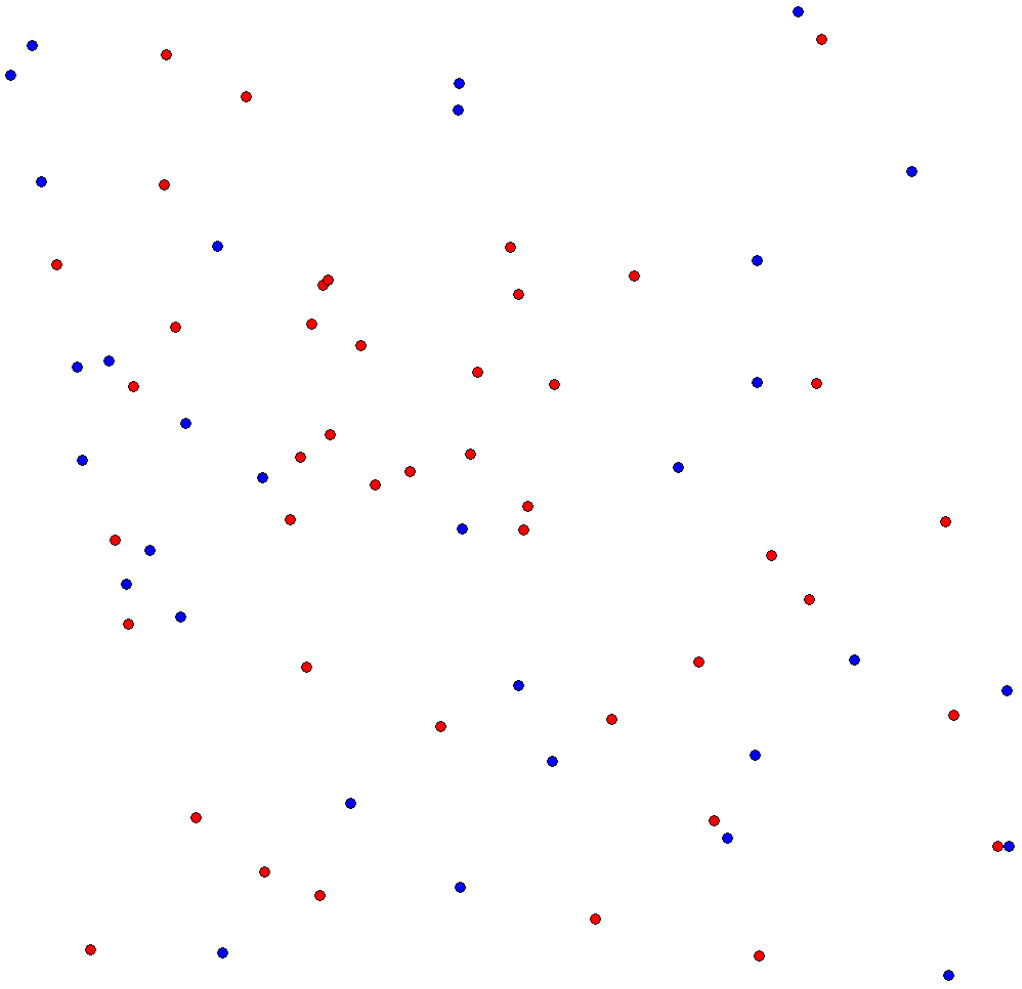
- 对于 Scheme2 数据，数据生成方式为生成一条斜率为 k 的直线以及一条斜率为 0 的直线，并且在平面上随机生成 **n** 个点，其中落在直线划分出的不同对角半平面的点分属不同集合



图例:scheme2 类型的数据点

算法\n	200	500	1000	2000	5000
Balanced	6.18	82.39	648.34	5191.78	-
Greedy	2.47	9.23	33.55	157.62	1351.07
PTS	3.53	14.96	48.90	204.99	1595.43

- 对于 scheme3 数据，数据生成的方式为在平面上随机生成 n 个点，每个点随机分属其中一个集合



图例:scheme3 类型的数据点

算法\n	200	500	1000	2000	5000
Balanced	6.45	83.63	657.73	5243.96	-
Greedy	15.76	166.44	1236.80	10052.59	-
PTS	132.52	2250.30	-	-	-

5. 小组分工

陈家浩：

- Balanced Separator 后半部分[论文[2]3.2(D) – (F)]
- PTSSeparator 后半部分[论文[2]5.2.2(D).c – (D).F]
- 算法的性能测试
- 算法演示部分的 Video 以及 Manual 的撰写
- 汇总所有代码生成最后的可执行工程

王翔：

- 实现 K-Enclosing 最小 k 包围圆算法[论文[1]]
- Balanced Separator 前半部分[论文[2]3.2(A) – (C)]
- 实现了基于 K-Enclosing 的λ-dense set 的分割集[论文[2] 5.2.2(D).a Theorem 5.2 - Appendix A.2 (A)-(G)]
- 报告-算法说明的问题一描述撰写
- 报告-数据结构与模块设计中的 k-包围圆部分撰写

张显觉：

- 实现 Voronoi Diagram 生成
- 实现了论文中的 GreedySeparator 算法[论文[2]5.1.1 5.2.2(A)]
- 实现了 GUI，功能包括自动生成数据，手动在平面上点出数据，调用其它接口生成分割图等
- 报告-数据结构与模块设计中 Voronoi 图算法报告, 输入输出接口的制定以及 GUI 的绘制
- CMakeLists 工程构建，适用于 Linux 下进行工程编译

6. 实验总结

- ✧ coding style 很重要，尤其是和其他人合作写 code，接口的设计好坏直接影响到算法的实现效率
- ✧ 写 code 之前准备工作做的越充分，后续 debug 的频次就会越低
- ✧ 合作 coding 需要设计好接口、输入输出格式、做好模块化设计，如此每个人可以放心实现自己的部分，后期各自模块无缝拼接起来
- ✧ 重构代码时思考了如何简练代码、增加可读性
- ✧ 对于算法中的计算 l-dense set，很多时候算法使用了 grid 技巧，也就是使用正方形网格来采样。但是作者使用的正方形网格的 sidelength 对于一般情况下的图太致密了，导致 separator 很大。此时适当增加 sidelength 可以非常有效地减少 separator 的大小，并且依然满足将两个集合分开的条件。
- ✧ 对于性能测试来说，两个集合的混乱程度越高，就需要越多的 separator，而且此时的算法开销增长非常快。所以选用不同的数据分布对算法有至关重要的性能影响。

7. 参考文献

[1] Alon Efrat, Micha Sharir, Alon Ziv. *Computing the smallest k-enclosing circle and related problems. Computational Geometry 4.3(1994): 119-136*
[2] V. V.S.P. Bhattiprolu, S. Har-Perld. Separating a Voronoi Diagram via Local Search. In Proc. 32nd Annu. ACM Sympos. Comput. Geom. (SoCG), pages 18.1-18.16, 2016.