无线传感网节点覆盖问题研究

王胤 应圣钢 刘青伟

2010-12-26

目　录

1．背景介绍

无线传感网技术成为 2010 年最热门的技术之一，无线传感网将传统分离的物理世界和信息空间实现互连和整合，代表未来网络的发展趋势，可广泛应用于森林火灾预警、地震救援、环境监测、军事目标监测、智能交通等领域。作为普适计算和自组织网络技术的延伸，无线传感网技术要解决的一个重要问题就是节点覆盖问题。在典型的森林火灾预警案例中，部署大量的低成本的传感器节点，通过自组织的方式无线相连，如何评估无线传感网的覆盖性能，也就是说这个无线传感网发现着火点的概率如何，是一个亟待解决的问题。 通过评估覆盖性能，可以找到侦测盲区，为进一步优化系统覆盖，提高无线传感网侦测服务质量(QOS)提供决策支持。

在无线传感网节点覆盖问题研究中，存在两种看起来截然相反但实际上非常相关的问题，即最差和最优节点覆盖。最差节点覆盖问题研究的是找到侦测性能较低的区域，以找到合适的路径，防止被无线传感器网络侦测到。最优节点覆盖问题研究目标为找到侦测性能较高的区域，以找到合适的路径，让无线传感器网络提供更好的支持服务和引导。最差节点覆盖的一个实际应用就是分布式雷达基地对敌方飞机的侦测问题。敌方飞机若想穿越分布式雷达基地的责任区域，应尽量沿着不被雷达发现的路径飞行。最优节点覆盖的一个实际应用野外探险问题。在给定的一片待探索区域中，预先布撒一些信标点。在小分队野外探险的过程中，应尽量贴近信标点，以保持与指控中心的联络和安全。

本文拟对这两类节点覆盖问题进行研究，结合计算几何的相关理论和算法，试图达到如下目标：

1) 深入学习体会基于 Beachline 的 Voronoi 图和基于随机增量 Delauny 三角剖分的构造算法；
2) 将计算几何的知识应用于实际无线传感器网络背景中，体会实际问题对基础理论的应用需求，以及综合计算几何和图论等理论算法解决问题的思路；
3) 在学习前人工作的基础上，提出一些改进的方法或思路。

2．问题描述

2.1 最差覆盖问题

对于最差覆盖问题，可以进行如下定义：

给定区域 A，以及在区域 A 中固定或者部署的无线传感器节点集合 S，对于任意无线传感器节点 $s_i \in S$，假设其位置点 (x_i, y_i) 已知。指定区域的起点 $I(x_I, y_I)$，终点 $F(x_F, y_F)$ 为飞机进入区域的进入点和飞出点，问题是找到一条路径 P_B，对于其上任意一点 $p \in P_B$，p 到其相邻的传感器节点 $\{s_a, s_b, ...\}$ 的距离始终保持最大。

由计算几何的知识可知，到相邻的传感器节点 $\{s_a, s_b, ...\}$ 的最小距离最大的点一定位于

相邻传感器节点$\{s_a, s_b, \ldots\}$的 Voronoi 图的公共边上，如图 1 所示。然而，即使沿 Voronoi 图的公共边进行飞行，仍然存在不同的路径满足从进入点飞入，从飞出点移出。因此需要定义不同的评价函数，来选定哪条路径在给定的评价函数意义上，符合最优性条件。

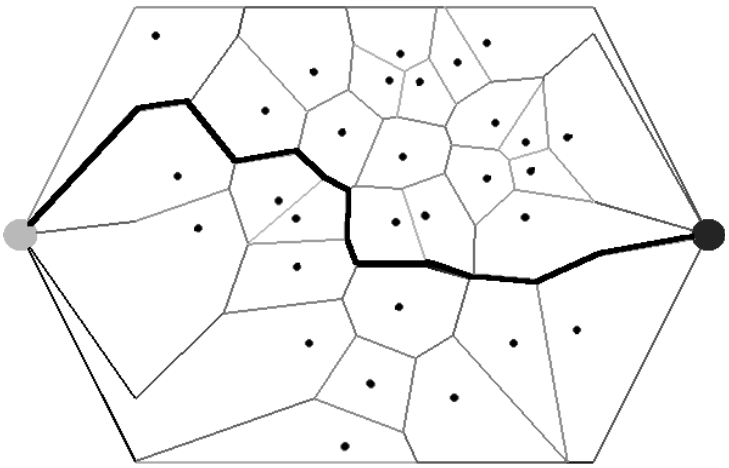


图 1 最差覆盖问题的 Voronoi 图解决方法(左边为进入点，右边为飞出点)

1) 评价函数 1-最小权重最大化

在该评价函数中，为每条 Voronoi 边指定一个权重，权重值为相邻两个节点到该边的距离。评价函数定义为对于任意一条出发点到飞出点的路径P_{Bj}，首先找到其包含的所有 Voronoi 边的最小权重$\min_{l\in P_{Bj}}\{weight(l)\}$，定义为路径$P_{Bj}$的权重，然后找到所有路径权重的最大值，即满足

$$\max_{P_{Bj}} \min_{l\in P_{Bj}} \{weight(l)\}$$

的路径P_B。

2) 评价函数 2-综合危险最小化

评价函数 1 可能在寻找路径时，为了达到优化目标，绕了很多弯路。一个自然的想法是能否在某些局部作出牺牲，只增加一点的危险系数（即最小权略变小），但可以使路径达到更好的全局效果，即走过的路径变少，经历过的传感器变少。综合考虑后，我们定义评价函数 2，即综合危险最小化。我们把一条 Voronoi 边上的危险系数统一定义为它的权的倒数，此条边上的危险总和就简化为长度乘以危险系数。问题就变为找到综合危险最小的路径，即满足

$$\min_{P_{Bj}} \left\{ \sum_{l\in P_{Bj}} \frac{length(l)}{weight(l)} \right\}$$

的路径P_B。

可以看出，这个评价函数可以比较直观的反应经过路径的综合危险最小化。同时，还可以一定程度上兼顾评价函数 1 中距离最近传感器最远的要求。

3) 评价函数 3-阈值权路径最短

评价函数 3 假定的场景是在满足一定危险性的条件下(即路径权重大于指定阈值权重)，

飞行航路路径长度最短。即满足

$$\min_{\{P_B|\forall l\in l\in P_{Bj},weight(l)>w\}}\left\{\sum_{l\in P_{Bj}} length(l)\right\}。$$

的路径 P_B。

这个评价函数在实际场景中具有重要的应用。飞行员在规划任务航路的时候，总是假定在满足一定的雷达侦察概率的条件下找到最优的路径。

2.2 最优覆盖问题

对于最优覆盖问题，可以进行如下定义：

给定区域 A，以及在区域 A 中固定或者部署的无线传感器节点集合 S，对于任意无线传感器节点 $s_i \in S$，假设其位置点 (x_i, y_i) 已知。指定区域的起点 $I(x_I, y_I)$，终点 $F(x_F, y_F)$ 为飞机进入区域的进入点和飞出点，问题是找到一条路径 P_B，对于其上任意一点 $p \in P_B$，p 到其相邻的传感器节点 $\{s_a, s_b, ...\}$ 的距离始终保持最小。

这个问题理解起来稍微有些复杂。如果飞机想穿越区域 A，假设其出发点 $I(x_I, y_I)$ 在某个传感器节点 $s_i \in S$ 负责的 Voronoi 区内，则该飞机首先第一个目标必须飞向节点 s_i，这样才能保持到 s_i 的距离最小，特别的，在 s_i 处的最小距离为 0。在飞向节点 s_i 的所有路径中，直线路径最短。当飞机在 s_i 处时，它必须决定下一个目标点，下一个目标点一定要负责 s_i 节点所对应得 Voronoi 区中相邻的一个 Voronoi 区，否则飞机在穿越邻居 Voronoi 区时，就必然存在一个不能到相邻的传感器节点距离最近的一条路径。因此，飞机下一个目标点一定是无线传感器节点集合 S Delaunay Triangulation 中与 s_i 具有公共边的节点，比如该目标节点设为 s_k，同样，在飞向节点 s_k 的所有路径中，直线路径最短。从上面的简要分析可知，飞机的任意一条穿越路径 P_{Bj} 必须满足沿着无线传感器节点集合 S Delaunay Triangulation 的边飞行，如图 2 所示。同样需要定义不同的评价函数，来选定哪条路径在给定的评价函数意义上，符合最优性条件。

图 2 最优覆盖问题的 Delaunay Triangulation 解决方法(左边为进入点，右边为飞出点)

4) 评价函数 4-最大权重最小化

在该评价函数中，为每条 Delaunay Triangulation 边指定一个权重，权重值为相邻两个节点到其公共 Voronoi 边的距离。评价函数定义为对于任意一条出发点到飞出点的路径 P_{Bj}，首先找到其包含的所有 Delaunay Triangulation 边的最大权重 $\max_{l\in P_{Bj}}\{weight(l)\}$，定义为路径 P_{Bj} 的权重，然后找到所有路径权重的最小值，即满足

$$\min_{P_{Bj}}\max_{l\in P_{Bj}}\{weight(l)\}$$

的路径 P_B。

5) 评价函数 5-综合支撑最大化

与评价函数 2 同样的思路，我们把一条 Delaunay Triangulation 边上的支撑系数统一定义为它的权的倒数，此条边上的支撑总和就简化为边长度乘以支撑系数。问题就变为找到综合支撑最大的路径，即满足

$$\max_{P_{Bj}}\left\{\sum_{l\in P_{Bj}}\frac{length(l)}{weight(l)}\right\}$$

的路径 P_B。

可以看出，这个评价函数可以比较直观的反应经过路径的综合支撑最大化。同时，还可以一定程度上兼顾评价函数 4 中距离最远传感器最近的要求。

注意：在本系统中，危险系数和支撑系数其实定义是一致的，考虑不同的问题，这里分开描述，下同。

3．系统实现

3.1 算法

3.1.1 基于 Beachline 的 Voronoi 图生成算法

本实验 Voronoi 图生成算法采用的是扫描线算法。VD 图生成部分的基本程序架构、生成算法采用的是前人《L-线段 Voronoi 图的扫描线算法的图形演示》程序，在他们的算法基础上，我们主要在 Voronoi 图生成上完成了如下工作：

1) 实现了 Voronoi 图的文件读写以及测试点的随机生成

在原程序中，测试点必须通过人工输入。本程序实现了测试点可通过界面随机产生，并且能够保存成文件，测试数据也可以通过文件读入程序，方便了系统测试。

2) 修改了相同 X 坐标输入点无法生成 VD 图的 BUG

在原程序中，当输入点的 X 坐标相同时，无法生成 VD 图。找到该问题后，我们通过将原始输入点序列以最左下角的输入点为原点，逆时针旋转一个极小的角度，使得在满足精度范围内，能有效产生 Voronoi 图，如图 3 所示。

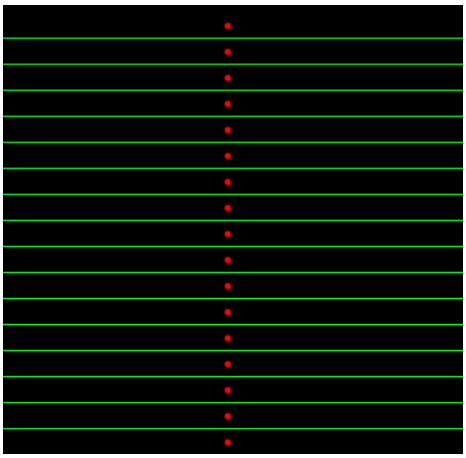


图 3　X 坐标相同时 VD 图生成的退化情况

3.1.2 基于随机增量的 Delaunay 三角剖分算法

1) 基于随机增量的 Delaunay 三角剖分算法简介

基于随机增量的 Delaunay 三角剖分算法的主要思路为：首先用一个足够大的三角形将整个点集 P 包围起来。为此需要引入两个辅助点 p_{-1}， p_{-2}，它们与 P 中的最高点 p_0 联合构成的三角形将包含所有的点。也就是说，我们计算的是 $P\bigcup\{p_{-1},p_{-2}\}$ 的（而不是 P 的）Delaunay 三角剖分，如图 4。为此，我们所选择的 p_{-1} 和 p_{-2} 必须相距足够远，才不致于对 P 的 Delaunay 三角剖分中的任何三角形有所影响，尤其必须保证的一点是，它们不能落在 P 中的任何三点的外接圆内。为了实现这一点，分别取位于整个点集 P 之下、之上的一对水平

线l_{-1},l_{-2}。假想地在l_{-1}上取p_{-1}，沿l_{-1}向右移动p_{-1}，直到它不再落在P中任何三个点外接圆内，并使P中各点相对于p_{-1}的极角次序，与它们的字典序完全一致。然后，假想在l_{-2}上取p_{-2}，沿l_{-2}向右移动p_{-2}，直到它不再落在$P\bigcup\{p_{-1}\}$中任何三个点外接圆内，并使$P\bigcup\{p_{-1}\}$中各点相对于p_{-2}的极角次序，与它们的字典序完全一致。

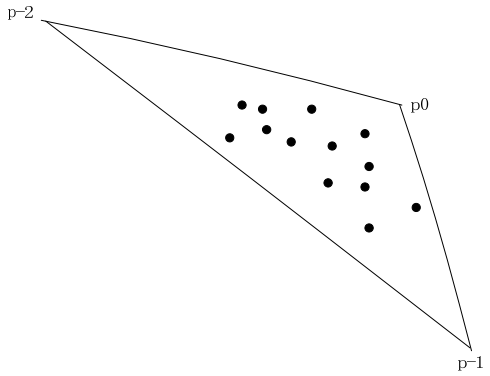


图 4. 包含点集P的三角形$\triangle p_0 p_{-1} p_{-2}$

随机增量法较为简单，遵循增量法的一贯思路，即按照随机的顺序依次插入点集中的点，在整个过程中都要维护并更新一个与当前点集对应的 Delaunay 三角剖分。考虑插入v_i点的情况，由于前面插入所有的点$v_1,v_2,...,v_{i-1}$构成的$DT\left(v_1,v_2,...,v_{i-1}\right)$已经是 Delaunay 三角剖分，只需要考虑插入v_i点后引起的变化，并作调整使得$DT\left(v_1,v_2,...,v_{i-1}\right)\bigcup v_i$成为新的 Delaunay 三角剖分$DT\left(v_1,v_2,...,v_i\right)$。在插入调整过程：首先确定落在哪个三角形中（或边上），然后将v_i与三角形三个顶点连接起来构成三个三角形（或与共边的两个三角形的对顶点连接起来构成四个三角形），由于新生成的边以及原来的边可能不是或不再是 Delaunay 边(即为非法边)，故进行边翻转来调整使之都成为 Delaunay 边，从而得出$DT\left(v_1,v_2,...,v_i\right)$。

2) 基于随机增量的 Delaunay 三角剖分算法伪码描述

Algorithm IncrementalDelaunay(P **)**
Input: 由 n 个点组成的二维点集 P

Output: Delaunay 三角剖分 $DT(P)$

1: add a appropriate triangle boudingTriangle $\triangle p_0 p_{-1} p_{-2}$ to P

2: random generate the insert sequence $p_1, p_2, ..., p_n$

3: **for** $i \leftarrow 1$ **to** n **do**

4: **Insert(** $DT(p_0, p_{-1}, p_{-2}, v_1, v_2, ..., v_{i-1})$, v_i **)**

5: **end for**

6: remove the boudingTriangle and relative triangles and edges

7: **return** $DT(P)$

图 5. Algorithm IncrementalDelaunay(P)

Algorithm Insert($DT\left(p_0, p_{-1}, p_{-2}, v_1, v_2, ..., v_{i-1}\right)$, v_i)

1: find the triangle $\triangle abc$ which contains v_i

2: **if** (v_i located at the interior of $\triangle abc$) **then**

3: add triangle $\triangle abv_i$, $\triangle acv_i$ and $\triangle cbv_i$ into DT

4: **LegalizeEge** $\left(DT, ab, v_i\right)$

5: **LegalizeEge** $\left(DT, cb, v_i\right)$

6: **LegalizeEge** $\left(DT, bc, v_i\right)$

7: **else if** (v_i located at one edge ab of $\triangle abc$ and $\triangle abd$) **then**

8: add triangle $\triangle acv_i$, $\triangle cbv_i$, $\triangle dbv_i$, and $\triangle dav_i$ into DT

9: **LegalizeEge** $\left(DT, ac, v_i\right)$

10: **LegalizeEge** $\left(DT, bc, v_i\right)$

11: **LegalizeEge** $\left(DT, db, v_i\right)$

12: **LegalizeEge** $\left(DT, da, v_i\right)$

13: **end if**

14：**return** $DT\left(v_1, v_2, ..., v_i\right)$

图 6. Algorithm Insert($DT\left(p_0, p_{-1}, p_{-2}, v_1, v_2, ..., v_{i-1}\right)$, v_i)

Algorithm LegalizeEge $\left(DT, ab, v_i\right)$

1: **if** (**isLegal** $\left(ab, v_i\right)$) **then**

2: remove edge ab , add new edge cv_i into DT (c is the third vertex in $\triangle abc$)

3: **LegalizeEge** $\left(DT, ac, v_i\right)$

4: **LegalizeEge** $\left(DT, bc, v_i\right)$

5: **end if**

图 7　Algorithm LegalizeEge$\left(DT,ab,v_i\right)$

3)　边的合法性检测

isLegal$\left(p_i p_j, p_r\right)$对于边的合法性检测，分四种情况：

- i, j 都为负数，即此时 $p_i p_j$ 为大三角形的边，则 $p_i p_j$ 肯定合法。

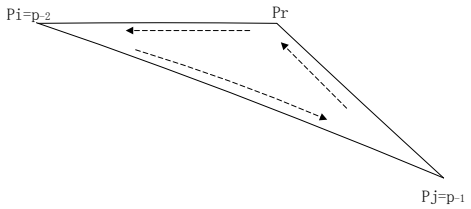


图 8. 合法性检测情况一

- i, j, k, r 都是非负数，这是一般的情况，若 p_r 位于 $p_i p_j p_k$ 的外接圆内，则非法；否则为合法边。

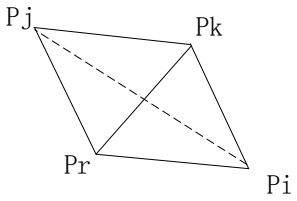
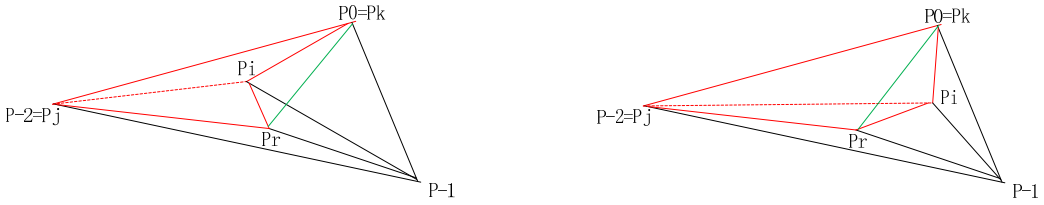


图 9. 合法性检测情况二

- i, j, k, r 中只有一个是负数。其中，r 为新插入的点，肯定不是负数。如果 k 为负数（k 为 $\triangle p_i p_j p_r$ 的邻接三角形 $\triangle p_i p_j p_k$ 的第三个顶点），则根据 p_{-1}, p_{-2} 的定义可知，p_{-1}, p_{-2} 一定落在另外三个顶点的外接圆的外面，因此这种情况合法；如果 j 为负数（i 为半边 $p_i p_j$ 的起点，j 为终点），这种情况按照教材上的观点（min(k, r) < min(i, j),则合法）一律是非法的，需要翻转的，然而这种情况也有一个反例如图 10(a)所示，如果翻转后将会出现相交与凹四边形。而同样的情况如图 10(b)所示，则可以正常翻转，针对这种情况我们对 $p_r p_i$，p_k 需要做一次测试，如果 p_k 在 $p_r p_i$ 逆时针方向，则翻转，反之则不翻转（包括共线的情况）；

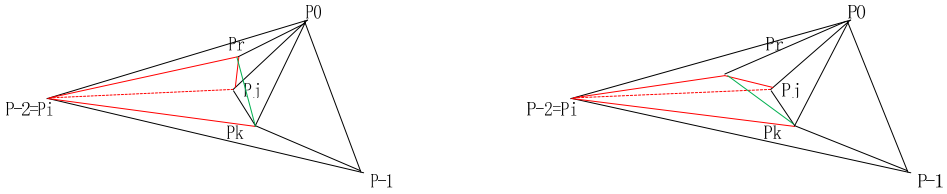


(a)　　(b)

图 10. 合法性检测情况三（ j 为负数的情况）

同理我们对仅有 i 为负数的情况也有类似的结论，对 $p_j p_r$， p_k 需要做一次测试，

如果 p_k 在 $p_j p_r$ 逆时针方向，则翻转，反之则不翻转（包括共线的情况）；



(a) (b)

图 11. 合法性检测情况三（i 为负数的情况）

- 其余的情况均合法，这些情况包括有两个下标为的负数的，一个为 k，另外一个为 i, j 之一。

4) 基于随机增量的 Delaunay 三角剖分算法复杂度分析

问题的规模为点集中的点的总个数 n（没有重合的点），循环内的基本的操作有：

a) 寻找插入点所在的三角形 `findBelongTriangle()`
b) 测试点是否在外接圆内 `InOutCircle()`
c) 更新三角网 （添加删除三角形，边，点）`Insert()` 和 `delete()`
d) 寻找共测试边的第三顶点
e) 翻转边 `LegalizeEge()`

考虑最坏的情况，时间复杂度为：

$$T = T\left(addBoundingTriangle(\ )\right) + Sum\left(T\left(Insert(i), i = 1,2,...,n\right)\right)$$
$$+T\left(removeBoundingTriangle(\ )\right)$$

因为 $addBoundingTriangle(\ )$ 和 $removeBoundingTriangle(\ )$ 不随 n 变化，是个常量，即 $T\left(addBoundingTriangle(\ )\right) = T\left(removeBoundingTriangle(\ )\right) = O(1)$。

因此时间复杂度为：$T = Sum\left(T\left(Insert(i), i = 1,2,...,n\right)\right) + O(1) + O(1)$。

考虑插入第 i 个的点的时间：

$$T\left(Insert(i)\right) = T\left(\text{findBelongTriangle}(i)\right) + T\left(UpdateDT(i)\right) + K \bullet T\left(\text{LegalizeEge}(DT, ab, v_i)\right)$$

（K 为常数，3 或者 4）

$$T\left(\text{findBelongTriangle}(i)\right)$$

要找出包含第 i 个点的三角形，由欧拉公式知道，平面图的面数 F 是 $O(n)$， n 为顶点

数，故此时总的三角形数是$O(i)$的。所以问题相当于在$O(i)$个记录中查找目标记录，如果不借助特殊的数据结构，按照一般顺序查找，需要$O(i)$的时间，即有

$T(\text{findBelongTriangle}(i))=O(i)$，故$Sum(T(\text{findBelongTriangle}(i)))=O(n^2)$。

这里我们采用了树形的查找存储结构，在下面的数据结构部分会有详细介绍。使查找目标结点的复杂度下降为$O(\log n)$，因此使得

$$Sum(T(\text{findBelongTriangle}(i)))=O(n\log n)$$

$$T(UpdateDT(i))$$

更新三角网数据结构，如果三角网（包括面，边，点）的数据是存入在一个双向链表的，插入和删除三角形到当前的三角网是个常量操作，因为已经知道插入和删除的位置。即有：

$T(UpdateDT(i))=O(1)$，故$Sum(T(UpdateDT(i)))=O(n)$

$$T(\text{LegalizeEge}(DT,ab,v_i))$$

比较复杂些，是个递归过程。细分为：

$$T(\text{LegalizeEge}(DT,ab,v_i))=T(FindThirdVertex(i))+T(InOutCircle(i))+$$
$$T(UpdateDT(i))+2*T(LegalizeEdge(j))$$

(这里，用 j 来区分不同的深度)

因为$InOutCircle(i)$是测试点是否在外接圆内，是个常量操作，不随点数变化而变化。故$T(InOutCircle(i))=O(1)$，又$T(UpdateDT(i))=O(1)$ (见上)。

$FindThirdVertex(i)$是查找目标点，在$O(i)$个记录中查找目标记录，如果不借助特殊的数据结构，按照一般顺序查找需要$O(i)$的时间。$FindThirdVertex(i)=O(i)$，由下面可知我们用到的是 DCEL 数据结构存储子区域划分，可以在常数时间内找到相邻三角形的第三个顶点，即有$FindThirdVertex(i)=O(1)$。

剩下的是递归调用$LegalizeEdge(\ )$的过程，不过还好，因为$LegalizeEdge(\ )$的递归深度只有常数级(设为 K)，正比于点在三角网中的度数(degree)，这是因为每一次的翻转都会使整个三角剖分的最小度数上升，而三角剖分是有限的，所以存在翻转边数的增长上限，是收敛的。所以 K 为常数。故$LegalizeEdge(\ )$最多调用的次数同样为常数。

故$T(\text{LegalizeEge}(DT,ab,v_i))=O(1)+O(1)+O(1)+O(1)=O(1)$，故

$$Sum\left(T\left(\text{LegalizeEge}(DT,ab,v_i)\right)\right)=O(n)$$

综上，最坏情况下算法总时间复杂度

$T=O(n\log n)+O(n)+K\bullet O(n)+O(1)+O(1)=O(n\log n)$。

另外，由于时间关系，部分数据结构采用原先同学的代码，使得部分实现的复杂度退化到了$O(n^2)$。

3.1.3 路径生成算法

1) 二分权查找的广度优先搜索

以评价函数 1 为例，二分权查找的广度优先搜索算法的思想是，首先设置一个可以接受的精度δ（即最终得到的结果与$\max\limits_{P}\min\limits_{l\in p}\{weight(l)\}$的差），从最小权与最大权的均值 w 开始，将权小于 w 的边去除，对留下的边作宽度优先搜索，查找是否存在一个可行的路径，如果有则以二分查找的方式将权变大，如果没有则变小，直到变化量小于初始设置的精度时停止算法。

此算法可以得到一个最小权至少为 w' 的路径，其中 w' 为算法最后一次搜到有解时的权。而最优解的权在区间$[w',w'+\delta]$内。

此算法的优点在于它在生成权大于某个权值 w 的图时，只需将所有边遍历一遍，而宽度优先搜索搜可行路径时，也只需线性世界，所以总的算法复杂度是$O\left(m\log\frac{W_{\max}-W_{\min}}{\delta}\right)$，其中，m 为边数。由于 Voronoi 图的边数目与初始点数目是线性关系，所以当计算的区域固定（即$W_{\max}$固定），精度δ固定时，算法复杂度可以为$O(n)$，算上 Voronoi 图的构造，整个计算过程的复杂度为$O(n\log n)$，其中 n 为探测器个数。

但该算法的缺点主要有：1）可能无法找到确定的解，因为最优解的全可能落在$(w',w'+\delta)$内，其中 w' 为算法最后一次搜到有解时的权；2）宽度优先搜索只能搜索到一条可行路径，只有当权均相等的时候，才能搜索到一条最佳路径，而这种情况基本不会发生。所以该算法只能应对评价函数 1 和评价函数 4 的情形，完全无法扩展，无非考虑其他优化因素。

二分权查找的广度优先算法如下所示。

```
Generate Bounded Voronoi diagram for S with vertex set U and
   line segment set L.
Initialize weighted undirected graph G(V,E)
FOR each vertex u_i ∈ U
   Create duplicate vertex v_i in V
FOR each l_i(u_j,u_k) ∈ L
   Create edge e_i(v_j,v_k) in E
   Weight(e_i)=min distance from sensor s_i ∈ S for 1≤i≤|S|
min_weight = min edge weight in G
max_weight = max edge weight in G
range = (max_weight – min_weight) / 2
breach_weight = min_weight + range
WHILE (range > binary_search_tolerance)
   Initialize graph G'(V',E')
   FOR each v_i ∈ V
      Create vertex v_i' in G'
   FOR each e_i ∈ E
      IF Weight(e_i) ≥ breach_weight
            Insert edge e_i' in G'
   range = range / 2
   IF BFS(G',I,F) is Successful
      breach_weight = breach_weight + range
   ELSE
      breach_weight = breach_weight – range
   END IF
```

在基于 Delaunay 三角剖分的三角网络上进行查找基于评价函数 4 的最大路径的生成算法，我们采用基于 BFS 的最优路径路径搜索算法，下面我们就对其复杂度进行分析。

由于 Delaunay 三角剖分的生成采用的是随机增量算法，因此期望的复杂度为 $O(n\log n)$；由于 Delaunay 三角剖分本身就是一张图，而且可以直接在其数据结构 DCEL 上进行宽度优先搜索（BFS），只需要小心的加入一些搜索控制条件即可，而且边的权重也可以在 $O(n)$ 的时间算出，因此并没有给整体的计算增加额外的开销。

在大部分情况下，BFS(包括 DFS)的复杂度为 $O(m)$（m 为图中边的数目），由于在实际的情况下我们处理的是稀疏三角网络，所以实际上的复杂度退化为 $O(n)$（n 为结点的数目），当然也可能达到 $O(n^2)$。在 BFS 的外层循环的次数是一个二分查找，可以在 *range* 对数时间内得到（而且这里我们暂且认为 *range* 与 n 成线性关系，可能更好）。因此，最坏情况下，该算法的时间复杂度退化为 $O(n^2\log n)$，而在 Delaunay 三角网络中，边的数目是与 n 成线性关系的，所以整体复杂度是 $O(n\log n)$。

2) Dijkstra 算法

以评价函数 2 为例，求满足 $\min_{P}\left\{\sum_{l\in P}\frac{length(l)}{weight(l)}\right\}$ 的最优路径，是一个标准的最短路径问题。可以用 Dijkstra 算法求解。由于 Voronoi 图边与顶点的线性关系，总的算法复杂为 $O(n^2)$。

对于评价函数 1，可以重新定义加法 $a+b:=\min(a,b)$，以及序关系反向"$\succ$"表示原来的 $<$，由于 min 函数对于路径的单调性，即 $\min(a,b)"\succ"a$，此情况满足一般 Dijkstra 算法的无负长度路径要求。仍可以用 Dijkstra 算法求解。

对于评价函数 4 和评价函数 5，由于是对偶问题，可以采用类似的分析，这里省略。

具体的 Dijkstra 算法参看一般的算法书。

3) 算法实现细节

求最大突破路径时，需要处理权的确定、起始路径、最终路径及其权的修正，处理 4 点共圆的退化情况。

a) 权的确定

计算某条边的权时，需要计算这条边上的点到某个 site 的最短距离。令此边两顶点到所属 site 的距离为 a 、 b ，边长为 c ，当 $c^2+a^2<b^2$ 或者 $c^2+b^2<a^2$ 时，site 到边的垂足不在边上，最短距离应为 $\min(a,b)$，否则最短距离应为边 c 对应的高长度。

b) 起始边、终止边的确定及权、危险系数(支撑系数)的修正

当起点、终点给定时，我们假设飞机只能在所给的区域内沿 Voronoi 图的边运动。而在起始、终止的情况，飞机可以沿左、右边界沿飞到某条 Voronoi 图边上。如图 12，黄色圈内点为起始终止点，飞机可在起始点、终止点在的两条黄线上平移到 Voronoi 图的边上。而和黄线相交且通向区域内部的边即可定义为起始（图中的红色边）、终止边（图中的紫色边）。白线为最优路径（评价函数 1）。

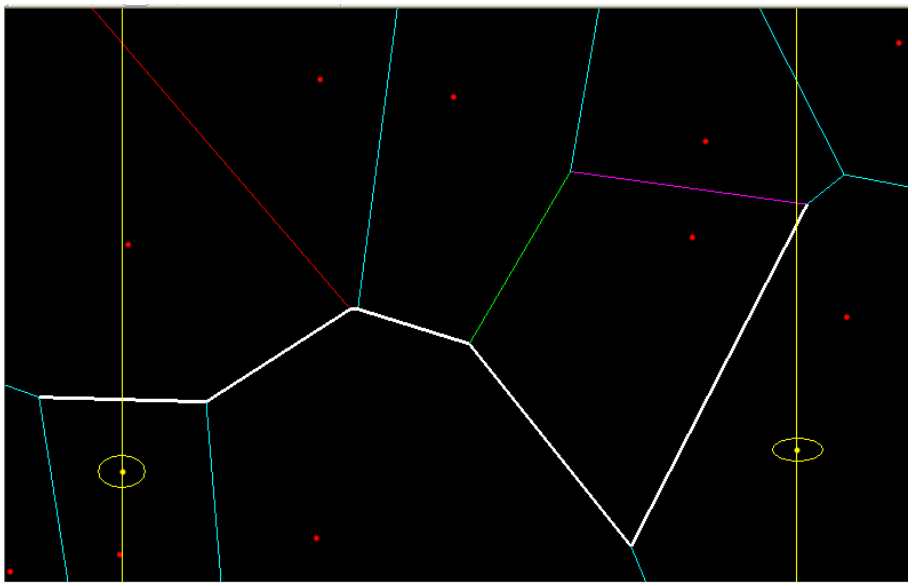


图 12 颜色标注说明

在确定起始边、终止边的权时，需要考虑黄线对其的影响。因为飞机并不飞满整个起始

终止边。同时，也需要考虑飞机从起点、终点沿黄线飞到起始、终止边所产生的权重、距离等。

c) DCEL 边的分类

除了 b）中提到的起始、终止边，在最优路径计算中，DCEL 边还可以分为无用边，即通向上下边界的边，因为我们定义飞机不能飞出区域外，上图中的浅蓝色边；顶点均在区域内的边为可行边，图中的绿色。

还有一种情况，起始边就是终止边，如图 13。图中的最佳路径就是一条既为起始边、又为终止边的 DCEL 边。

对这些边的分类，只需在 DCEL 边的结构中加一个参量即可（同时可表示颜色）。

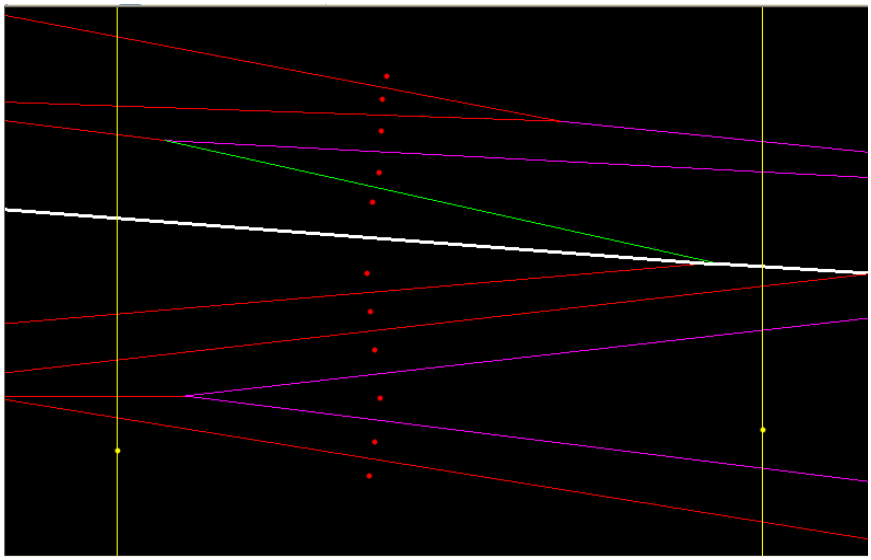


图 13 DCEL 边的分类

d) 四点共圆情况处理

四个 site 点共圆时，最终的 Voronoi 图会产生四条边共顶点的情况。对于这种情况，扫描线算法会产生一条长为 0 的边，将一个 4 度的顶点分为 2 个 3 度的顶点。一开始实验时，求这条长为 0 边的权时，所对应的情况为求其上的高，需要 2 被面积除以边。这样得到的结果是负无穷，导致这条路径是无法通过的（每次均判断权是否大于 0）。于是得到错误的结果（评价函数 1），如图 14。处理 4 点共圆的情况后，得到的正确结果（评价函数 1）如图 15。可以明显看出最优选择了离传感器节点最远的一条路径。

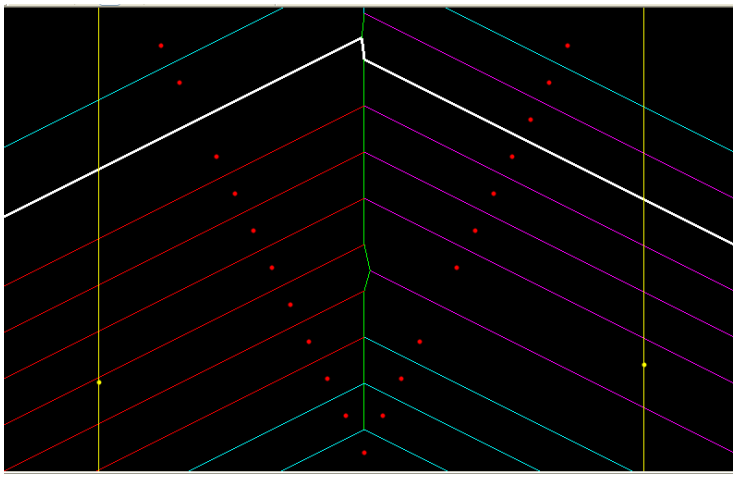


图 14 未考虑 4 点共圆时的错误结果（评价函数 1）

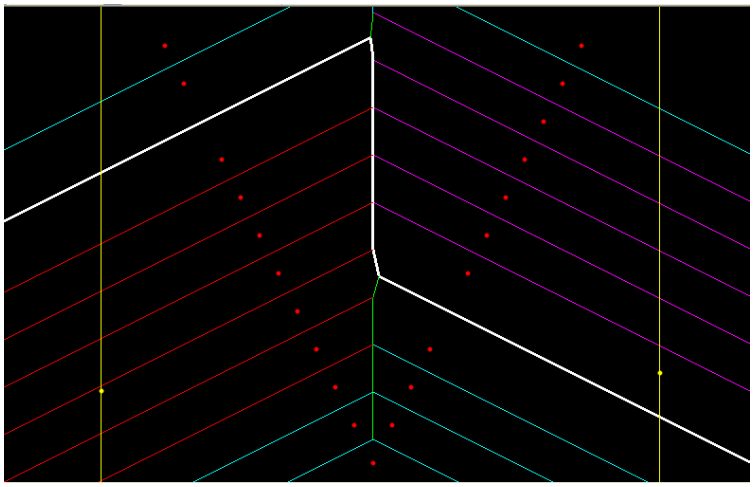


图 15 考虑 4 点共圆时的正确结果（评价函数 1）

e) 多最优路径时的情况

多最优路径时，由于评价函数 1 的性质，导致最后输出的最优路径可能在优化评价函数的结果上达到了最优，但可能绕远路（这些远路对评价函数 1 无影响，因为有更小的权限制）。

3.2 数据结构

3.2.1 DCEL 双向链接边表

DCEL 是存储平面子区域划分一种高效的数据结构，它不仅存储了子区域划分的点、边也存储了面。

DCEL 一个明显的特点就是将一条分成了两个半边 HalfEdge，每个半边都方向，互为反向，他们互为 twins，各有指针指向对方，并且还存储了起点，终点，包围的面，属于同一个面的下一个半边，上一条半边。数据结构可以用下面的代码来表示。其中面是由半边逆时针包围的（暂时不考虑中间有洞的情况）。

```
1 struct DCELHalfEdge{
2     DCELHalfEdge* twin;
3     DCELHalfEdge* prev;
4     DCELHalfEdge* next;
5     DCELFace* face;
6     DCELVertex* origin;
7 };
```

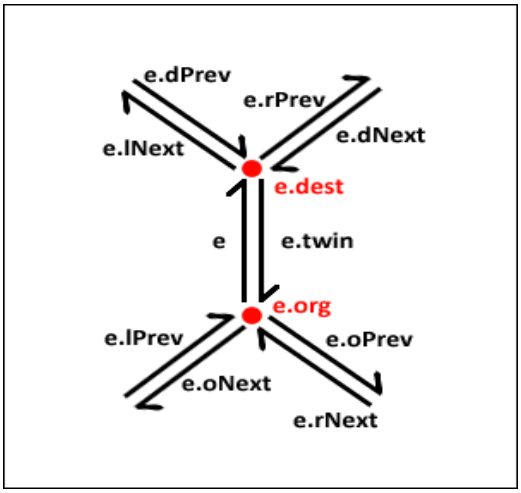


图 16. The Structure of DCELHalfEdge

DCEL 对于点的存储，只需记录点的坐标与一条以它为起点的半边即可。

```
struct DCELVertex{
  Coordinate coordinate;
  DCELHalfEdge *leaving;
};
```

DCEL 对于面的存储，只需记录围成该面一条半边即可。当然，他们的数据结构可以扩展存储其它需要的信息，这里只是存储了最基本的信息，如考虑到画图的对不同类型的边着色的需要，在 DCELHalfEdge 中我们还定义了颜色属性。

而对于面、点、边的访问（插入与删除）在常数时间内达到（见上面的算法分析），最好的存储方式是用**双向链表**。

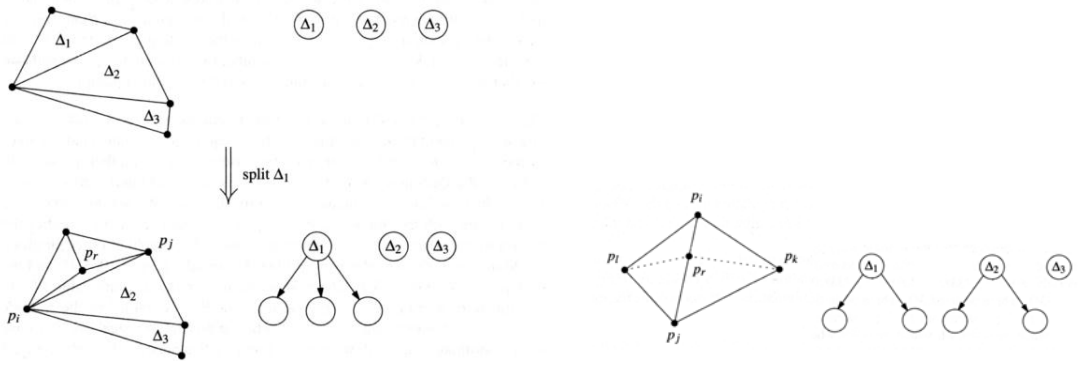
3.2.2 DAG 有向无环图（Directed Acyclic Graph）

上面算法分析可知，查找点所在的三角形 findBelongTriangle()，是一个复杂度的瓶颈。暴利的查找复杂度为 $O(i)$，使得系统整体时间复杂度为 $O(n^2)$，为此我们需要更为方便的数据结构。

DAG 是一个点定位结构，其中每片叶子（出度为 0 的结点）对应于当前三角剖分中一个三角面，而且互相有指针指向对方。而内部结点，都对应于曾经在些前某个阶段的三角剖分中存在过的某个三角形，只不过现在这个三角形已经被销毁了。

其构造过程有以下四个过程。

- 将 $\triangle p_0 p_{-1} p_{-2}$ 初始化为根结点。
- 如果新加入的点 p_r 将 $\triangle p_i p_j p_k$ 分为三个小三角形，DAG 中将会对应的增加三个结点作为 $\triangle p_i p_j p_k$ 对就结点的子结点（见图 8-a）。
- 如果新加入的点 p_r 将 $\triangle p_i p_j p_k$，$\triangle p_i p_j p_t$ 分别分为四个小三角形，DAG 中将会对应的在 $\triangle p_i p_j p_k$ 和 $\triangle p_i p_j p_t$ 对就的结点下分别增加两个子结点（见图 8-b）。
- 在 $\text{LegalizeEge}(DT, p_i p_j, p_r)$ 中，实现了翻转，则添加两个结点到翻转过程中需要删除的两个三角形 $\triangle p_i p_j p_r$,$\triangle p_i p_j p_k$ （k 为相邻三角形的第三个顶点）（见图 9）。



（a） (b)

图 17. DAG 构造过程

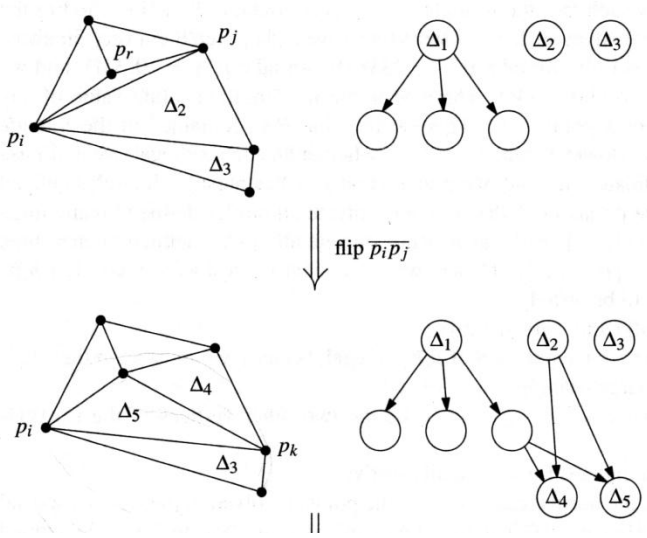


图 18. 在翻转操作过程的 DAG 结点构造

由于最后的 DAG 的叶子结点的数目为三角形的数目，由于总数目的期望值超过不 $9n+1$，所以可得树的高度为 $O(\log n)$。这样就可以在该 DAG 中在 $O(\log n)$ 进行点定位，找到所在三角形或边。

其简要代码如下所示：

```
struct PointLocationDAG{
    DCELFace *face;
    PointLocationDAG *childNode[3];
    PLDAGEdge edge[3];
};
```

其中，face 是指其对应的三角面（在三角面被删除时置空），childNode，是其子结点，最多有三个。edge 存入的是围成 face 所需要的三条边，由于 face 可能在增量过程中被删除，所以有必要将其保存下来。

在 DAG 中实现点的三角定位的伪代码如下，这是一个递归搜索的过程：

Algorithm findBelongTriangle$(p_r, node)$

1: **input:** p_r : DCELVertex, node: PointLocationDAG, pBelongFace: DCELFace,

2: edge: DCELHalfEdge

3: **switch** (**isBelong**$(p_r, node)$)

4: **case** NotBelong: **return** false

5: **case** Belong:

6: **for** $i \leftarrow 1$ **to** 3 **do**

7: **if** $node.childNode[i] <> NULL$ **then**

8: **if** findBelongTriangle$(p_r, node.childNode[i])$ **then**

9: **return** true

10: **end if**

11: **end if**

12: **end for**

13: pBelongFace $\leftarrow node.face$

14: **return** true

15: **case** OnEdge:

16: pBelongFace $\leftarrow node.face$

17: **return** true

18: **end switch**

其中，函数(**isBelong**$(p_r, node)$用于判断点 p_r 是否属于 $node$ 所在的三角形（也要处理共线的情况），只要验证点 p_r 是否是在 $node$ 所对应三角形三条半边的逆时针方向（即左侧）即可（可以根据坐标在常数时间内判别），这里要注意对应三角形的半边是大三角边情况，这个时候他们的相对位置关系，要根据 p_{-1}, p_{-2} 的定义来判断，即所有点的相对于他们的极角次序等于它们的字典序。

3.3 软件实现

本程序采用 VC2008 作为开发平台，用 MFC 作为应用程序框架，采用 OpenGL 作为底层图形库。MFC(Microsoft Foundation Classes)，是一个微软公司提供的类库（class libraries），以 C++类的形式封装了 Windows 的 API，并且包含一个应用程序框架，以减少应用程序开发人员的工作量。其中包含的类包含大量 Windows 句柄封装类和很多 Windows 的内建控件和组件的封装类。MFC 相对于 C#的 winform 与 Wpf 最大的优势在于其相对来说是面向操作系统底层的，在处理大量数据时候占有优势。

OpenGL（全写 Open Graphics Library）是个定义了一个跨编程语言、跨平台的编程接口的规格，它用于三维图象（二维的亦可）。OpenGL 是个专业的图形程序接口，是一个功能强大，调用方便的底层图形库。而且在 VC2008 环境下可以方便的进行 openGL 编程，在使用前先设置好编译环境，包含相应的头文件

#include <GL/gl.h>

#include <GL/glu.h>

#include <GL/glaux.h>

再添加相应的库文件，Project | Settings | Link | Object/library module | "opengl32.lib glu32.lib glaux.lib" | OK。

软件界面及按钮功能图如图 19 所示。

图 19 软件功能界面

软件运行流程图如图 20 所示。

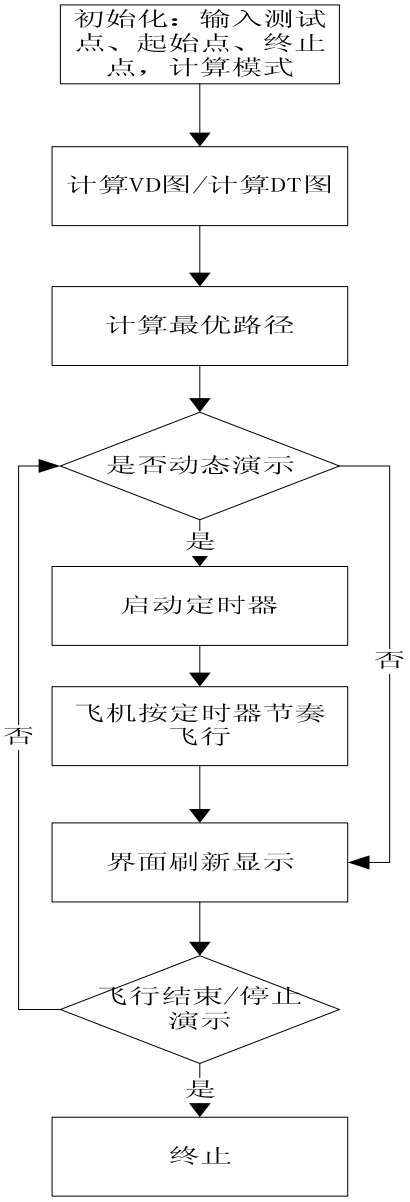


图 20 软件运行流程图

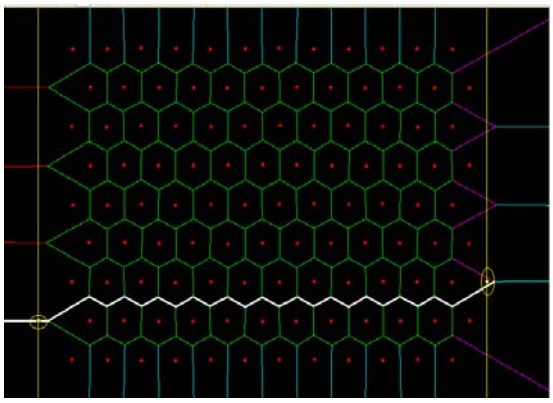
4. 结果分析

4.1 实验结果

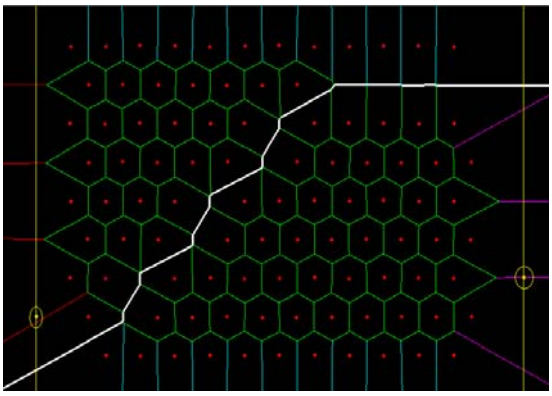
4.1.1 最差覆盖问题

1) 正确性验证

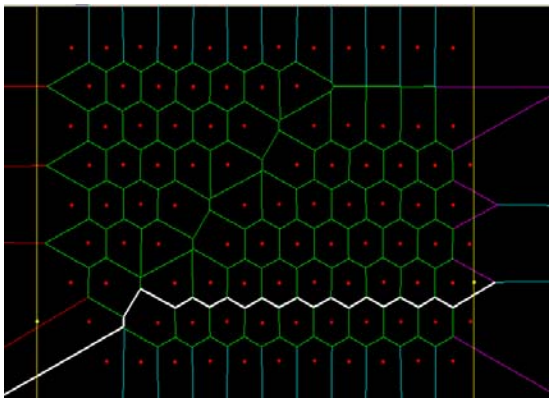
由于篇幅关系，我们仅以评价函数 1 为例，验证结果正确性。在正确性验证中，我们无法通过一般方法直观地显示，而数据显示正确性的方法并不可取，经过老师指点，我们设置了一些具有一定规律初始点图，以直观显示算法的正确性。



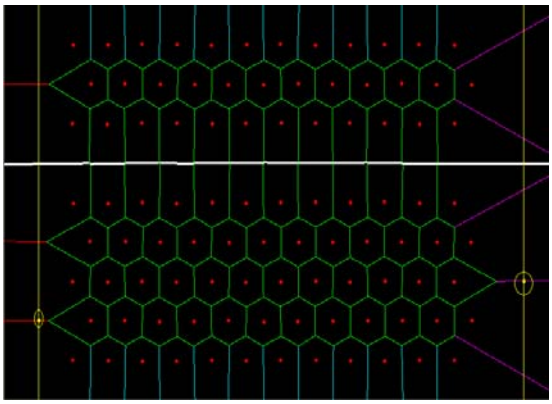
（a） 无坑时



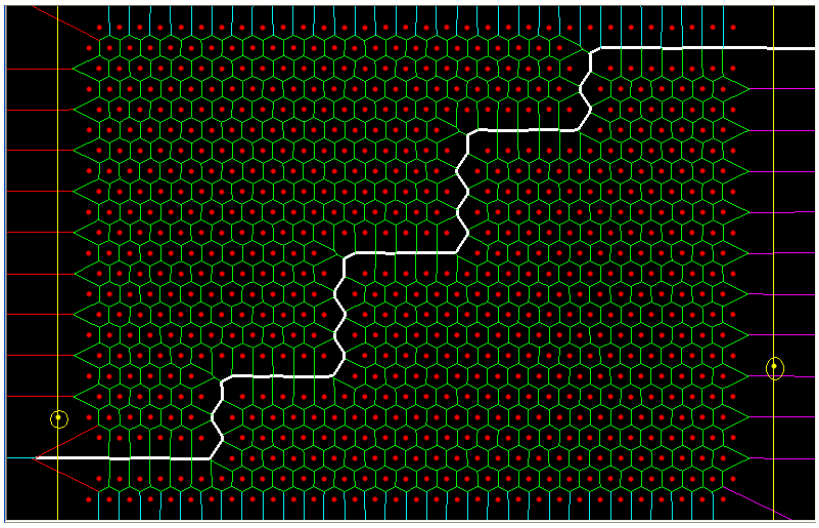
(b)有坑且点终止点所在竖线距离探测器较远



(c)有坑且点终止点所在竖线距离探测器较近



(d) 蜂房中间出现断层



(e) 750 点左右蜂房中间有梯形通道

图 21 蜂窝型测试点验证 VD 图正确性

图 21(a)给出了一个无坑（点全满）时最优路径。图 21(b)给出了有坑且点终止点所在竖线距离探测器较远情况时的最优路径。图 21(c)给出了同探测器分布下终止点靠近探测器时的最优路径，可以看到，由于终于点所在竖线靠近探测器，由竖线的权修正，最优路径发生了变化。图 21(d)显示了蜂房中间出现断层时的结果。图 21(e)显示了 750 点左右蜂房中间有梯形通道时的结果。

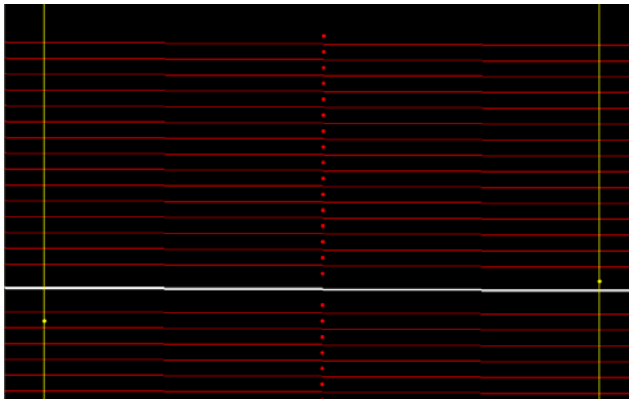
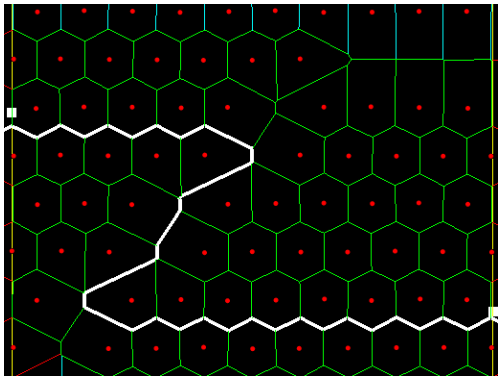


图 22 点列正确性测试

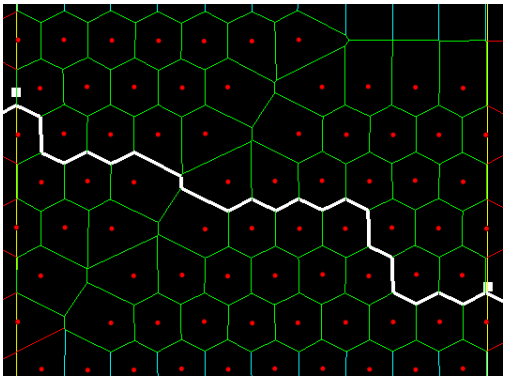
图 22 显示了点列中间有一个空隙使的情况。

从实验结果来看，据我们所能想到的测试序列和分析能力，结果是正确的。

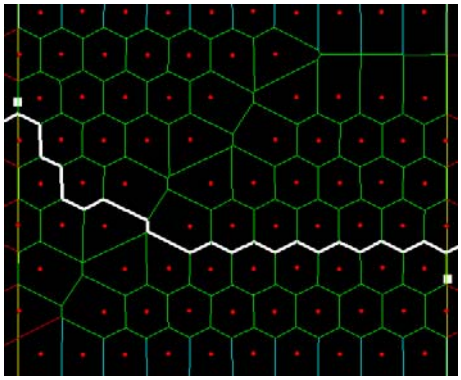
2) 不同评价函数下的实验结果



(a) 评价函数 1



(b) 评价函数 2



(c) 评价函数 3(指定权重 10.0)

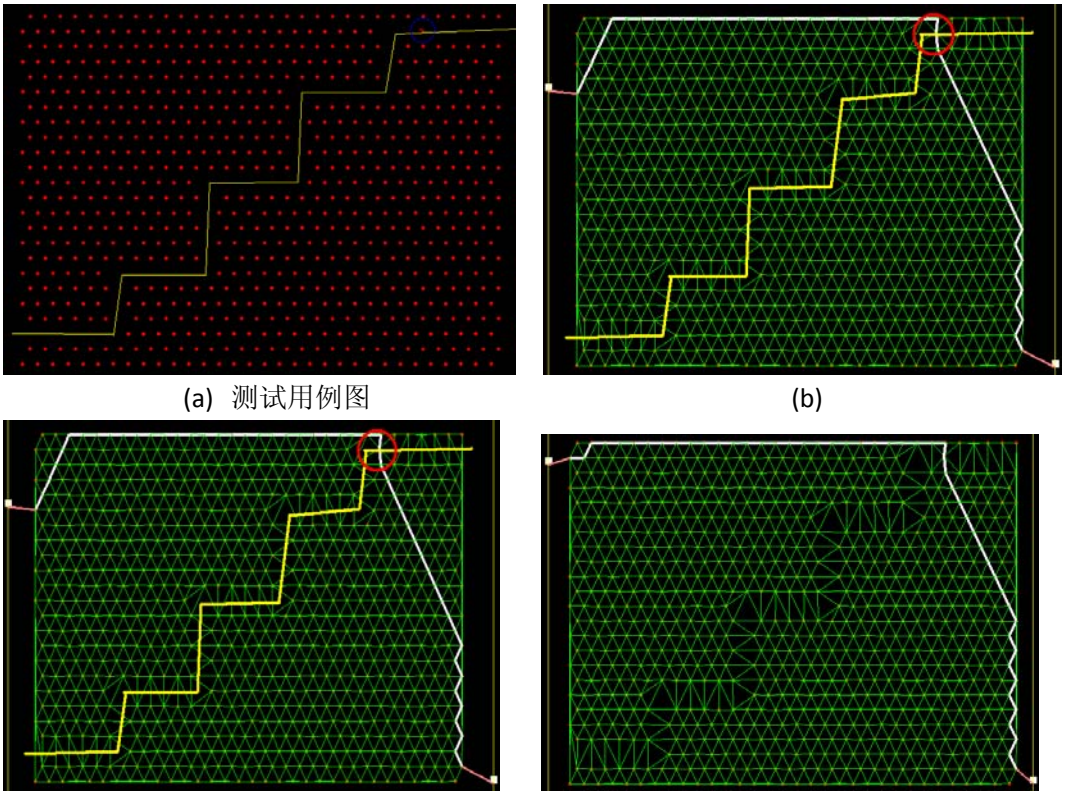
图 23 在不同的评价函数下实验结果对比

从实验结果可以直观的看出，评价函数 2 相对于评价函数 1 来说，路径明显减小。评价函数 1 仅为了寻找一个离传感器节点尽量远的路径，评价函数 2 在路径长度和传感器节点距离找一个折中。评价函数 3 给出指定权重为 10.0 的最优路径。

4.1.2 最优覆盖问题

1) 正确性验证

给定如下的测试用例：



(a) 测试用例图　　(b)

(c)　　(d)

图 24 “楚河-汉界”图验证 DT 图正确性

在上图的测试用例上，黄色的狭长地带是点与点之间相对距离较远的点，其中只有一个点（蓝圈所标示）相对距离较近，我们称为蓝点，可以认为黄色线是一条鸿沟，而蓝点是鸿沟上一座桥，如果想安全的（尽可能的贴近点航行）从上半部跨越到下半部航行，蓝点是必经的。(b)(c)(d)是在不同起始和终止点的情况下，可达的测试路径。可以看出所有的测试曲线都经过了蓝点。

2) ***不同评价函数下的实验结果***

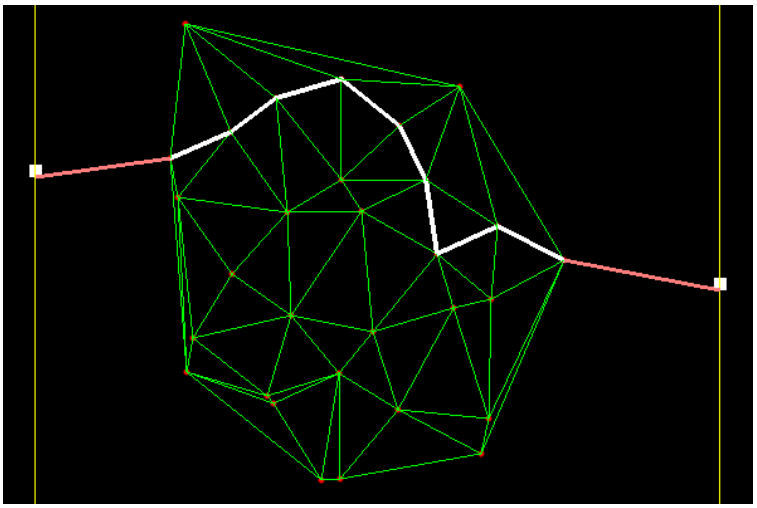


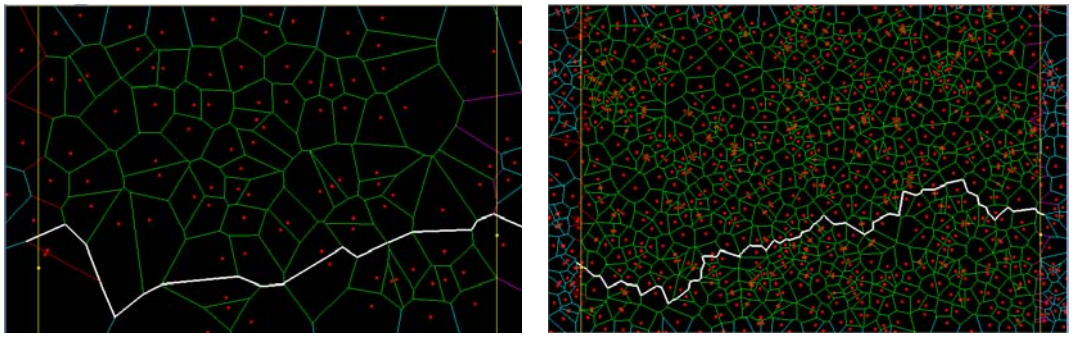
图 25 在评价函数 4 条件下，最优覆盖问题的实验结果

由于时间关系，在最优覆盖问题上，我们仅完成了一个评价函数。比较图 2 和图 25，可以看出，与文献[1]中的结果是基本一致的。

4.2 压力测试

在压力测试中，我们测试随机生成 100 个点、1000 个点在最差覆盖问题、最优覆盖问题两种条件下不同的测试结果。测试机器配置为双核 1.8Ghz、2.5GB 内存。

在评价函数 1 条件下，最差覆盖问题压力测试如图 26 所示。图 26(a)是 100 个随机生成点时的情况，运行时间不到 1s。图 26(b)是 1000 个随机生成点时的情况，运行时间大概 5 秒。

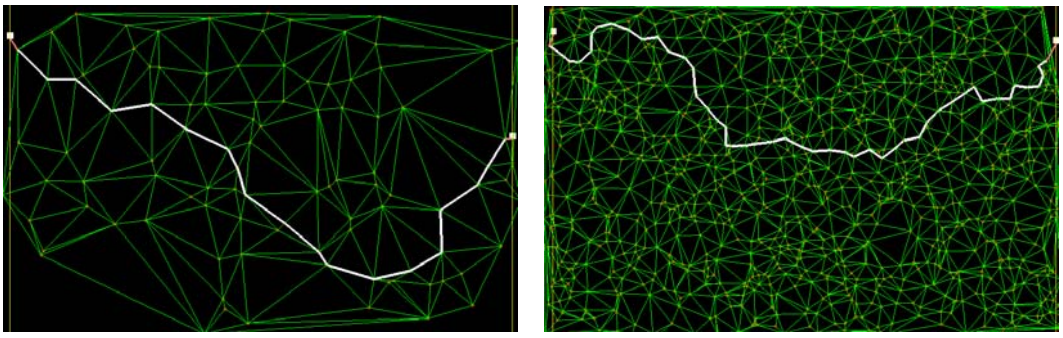


(a) 100 个随机生成测试点　　(b) 1000 个随机生成测试点

图 26 在评价函数 1 条件下，最差覆盖问题压力测试

在评价函数 4 条件下，最优覆盖问题压力测试如图 27 所示。

图 27(a)是 100 个随机生成点时的情况，运行时间不到 100ms。图 27(b)是 1000 个随机生成点时的情况，运行时间大概 1.5 秒。



(a) 100 个随机生成测试点　　(b) 1000 个随机生成测试点

图 27 在评价函数 4 条件下，最优覆盖问题压力测试

5. 问题及进一步工作

5.1 实验中发现的问题及解决情况

虽然整个实验参照文献[1]展开，但在实验过程中，在前人的基础上，我们还是有不少体会，主要列举如下：

1) **发现教材[2]中基于随机增量的 Delaunay 三角剖分中关于边的合法性检测存在问题。**按照教材的观点，当 j 为负数时，需要翻转，然而翻转后会出现相交与凹四边形。详细的分析见第 3.1.2 章节相关内容。**我们同时提出在此种情况下的翻转策略；**

2) **与文献[1]相比，分析基于 BFS 的路径生成算法的复杂度是 $O(n)$ 而不是**

$O(n\log n)$。在文献[1]中，所提出的基于 BFS 的路径生成算法的复杂度是$O(n\log n)$，其理论基础在于 BFS 算法的复杂度是$O(m)$（m 为图中边的数目），一般来说，DT 图和 VD 图的边数与 n 成线性关系，BFS 外层循环是二分查找，可以在 *range* 对数时间内得到(文献假定 *range* 与 n 成线性关系)。但进一步的分析我们发现，给定区域范围内， *range* 与 n 无关，甚至有可能是负相关，所以 BFS 的路径生成算法的复杂度是$O(n)$，详细分析见第 3.1.3 章节；

3) **与文献[1]相比，提出二种新的评价函数 2 和 3。评价函数 2 综合考虑总体路径长度和离节点的距离，在综合情况下效果更优，评价函数 3 在实际背景中具有重要的意义。**实验结果验证了评价函数 2 的性能，见 4.1.1 (2)不同评价函数下的实验结果章节；
4) **在邓老师提出的实验结果无法正确性验证的问题上，我们独立构造了“蜂窝图”和“楚河-汉界”图。**从图上能够较为直观的验证实验结果，见实验结果 4.1.1(1)和 4.2.1(1)正确性验证章节；
5) **设计动态飞行演示验证数据结构。**在实验中，我们增加了飞机动态飞行演示功能，来更为直观的描述问题以及验证数据结构的“Live”特性；
6) **修改了相同 X 坐标输入点无法生成 VD 图的 BUG。**采用将初始坐标点随机旋转一个小角度的方法实现。本来想使用邓老师在课堂上讲授的“虚拟旋转”方法，从上到下处理 Site Event 事件，但由于整个基于 Beachline 的算法为使用《L-线段 Voronoi 图的扫描线算法的图形演示》算法，改动处较多，由于时间关系，没有使用“虚拟旋转”办法；
7) **运算精度问题。**在程序中，我们使用双精度数进行运算。在判断两个数是否相等以及随机旋转小角度时，使用了容差 1e-5 进行判断。

我们同时解决了在初步汇报时邓老师提出的一些问题，列举如下：

1) 关于起点和终点的界面直观显示，以及对起点、终点的处理；
2) 测试数据的随机产生；
3) 测试文件的读写；
4) 对问题的详细的描述和说明，见第 2 章节；
5) 对原始程序界面的修正；
6) 对无效区域的区别显示；
7) 增加退化性测试用例。对退化测试用例的处理见 3.1.1 章。

5.2 进一步的工作

由于时间关系，非常遗憾，我们仍然有一些工作未能实现。进一步的工作包括：

1) 在最优覆盖问题中，评价函数 5 的实现；
2) 将背景改为实际地形图，以使问题背景更直观；
3) 在节点权重不同的条件下，最差覆盖问题的定义及解决方法。

这涉及到加权 VD 图的问题。加权 VD 图是给普通 VD 图的每个节点附上权重，其定义如下：设二维欧氏空间上的一个离散点集$S=\{p_1,p_2,...,p_n\}$，由

$$V(p_i, w_i) = \underset{j \neq i}{I} \{p \mid \frac{d(p, p_i)}{w_i} < \frac{d(p, p_i)}{w_j}\}$$

定义的区域称为点 p_i 的加权 VD 多边形，所有 $S = \{p_1, p_2, ..., p_n\}$ 构成的加权 VD 多边形区域形成加权 VD 图。$d(p, p_i)$ 定义为欧氏距离，w_i 为节点 p_i 权重。当所有节点权重相等时，退化为 VD 图。

文献[3]中给出加权 VD 多边形区域是由若干圆弧公共边形成，设 L_{ij} 为节点 $p_i(x_i, y_i)$ 和 $p_j(x_j, y_j)$ 公共边，L_{ij} 的圆心为 O，则有如下性质(假设 $w_i < w_j$)：

1) 圆心坐标为($\frac{w_i^2 x_i - w_j^2 x_j}{w_i^2 - w_j^2}, \frac{w_i^2 y_i - w_j^2 y_j}{w_i^2 - w_j^2}$);

2) 半径 $R = \frac{w_i w_j d(p_i, p_j)}{\left|w_i^2 - w_j^2\right|}$

3) $d(O, p_i) = \frac{w_i^2 d(p_i, p_j)}{\left|w_i^2 - w_j^2\right|}, d(O, p_j) = \frac{w_j^2 d(p_i, p_j)}{\left|w_i^2 - w_j^2\right|}$

O, p_i, p_j 共线，且 p_i, p_j 关于 O 互为镜像点。

典型的加权 VD 图如图 28 所示。

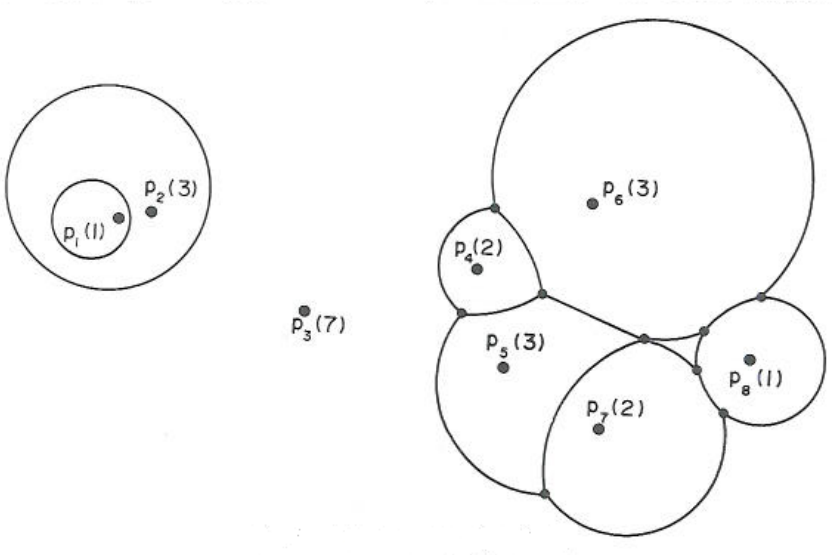


图 28 典型的 8 个点的加权 VD 图[3]

如图 28 所示，由于在加权 VD 图条件下，当节点之间的权重相差比较大时(>20%)，很容易不存在封闭的连通路径，所以在最差覆盖问题中的所有评价函数均无法使用。

文献[3]提出一种 $O(n^2)$ 的 WVD 构建方法，文献[4]提出一种随机增量的 WVD 构建方法，但未给出算法的复杂度。在实际应用中，节点权重描述的是节点侦测能力的**差异性**。由于制

造条件的差异性，不可能制造出完全相同的节点，所以节点侦测能力存在一定的差异，但这种差异一般最大不会超过 5%，当节点分布较为均匀时，这样构造出来的加权 VD 图仍然存在连通路径。一个自然的想法在权重误差小于 5%的条件下，能否采用近似的方法构造加权 Voronoi 图呢？

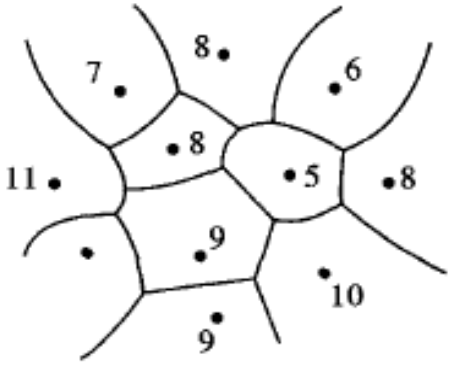


图 29 权重相差不大的 WVD 图

在该种情况下，可以使用一个线段公共边来代替圆弧公共边 L_{ij} 。线段公共边(或延长线)与两个节点连线的交点坐标为($x_i + \frac{w_i}{w_i + w_j}(x_j - x_i), y_i + \frac{w_i}{w_i + w_j}(y_j - y_i)$)。一个初步的思路能否基于扫描线算法构建此种近似情况下的 WVD 图呢？由于时间关系，该算法未能够实现，将在后期的工作中完善。

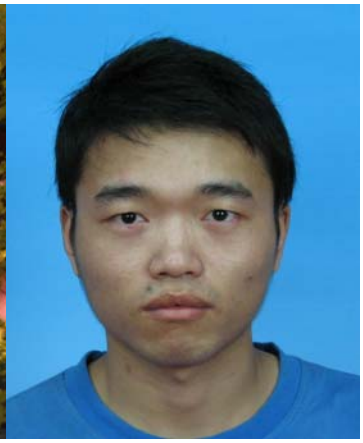
6．结论

无线传感网技术将传统分离的物理世界和信息空间实现互连和整合，代表未来网络的发展方向。无线传感网的覆盖性能是研究其服务质量的重要指标。本课题组研究了最差和最优覆盖问题，结合计算几何和图论的相关知识，提出了基于 Voronoi 图、Delanay 三角剖分以及 BFS 的路径搜索算法。本课题组的主要工作包括：

1) 独立实现了基于随机增量的 Delaunay 三角剖分算法，并指出教材中关于边的合法性检测中存在的问题；
2) 基于《L-线段 Voronoi 图的扫描线算法的图形演示》的源代码，完成了基于 BFS 和 Dijkstra 的路径生成算法；
3) 提出并实现了 5 种评价函数，其中评价函数 2 实际运行效能优于文献[1]的最小权最大化算法；独立构造了测试样本，验证了评价函数的正确性；
4) 设计了飞行动态演示，提高了待研究问题的直观性，验证了数据结构；
5) 学习了加权 VD 图的基本构造方法，**初步**提出了在近似条件下加权 VD 图的构造思路。

7 团队联系方式：





王胤	应圣钢	刘青伟
Wangyin00@gmail.com	yingsg@yahoo.com.cn	592921184@qq.com

参考文献：

[1] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava, “Coverage Problems in Wireless Ad Hoc Sensor Networks,” Proceedings of IEEE INFOCOM, April 2001.

[2] Mark de Berg, etc.著，邓俊辉译，“计算几何-算法与应用”，清华大学出版社.

[3] F.Aurenhammer, etc. “An Optimal Algorithm for constructing the weighted voronoi diagram in the plane”, in *Pattern Recognition*, Vol. 17, No. 2, pp. 251-257, 1984.

[4] F.Aurenhammer, “Voronoi Diagrams- A survey of a Fundamental Geometric Data Structure”, in ACM *Computing Surveys*, 1991.