维普资讯 http://www.cqvip.com

近似线性平均复杂性的平面点集 Voronoi 图增量算法的设计与实现*)

Design and Implementation of an Approximately Linear Average-Case Complexity Algorithm for Voronoi Diagram of Planar Point Set

廖士中[1] 王晓东[1,2]

(辽宁师范大学计算机与信息技术学院 大连116029)[1] (牡丹江师范学院物理系 牡丹江157000)[2]

Abstract This paper designs and implements a valid and efficient incremental algorithm for Voronoi diagram of planar point set. Based on the Winged Edge Data Structure, the algorithm adopts the technique of bucketing to select a generator and to speed up nearest neighbor search process, and can deal with the degradation cases, such as co-linearity of three points and co-circularity of four points, and computation errors. Although the theoretic time complexity of the algorithm is $O(n^2)$, the experiment results show that the average-case time complexity is $O(n)$ approxemately.

Keywords Computational geometry, Voronoi diagram, Incremental algorithm, Bucketing

1 引言

Voronoi 图是计算几何学科的一个重要结构,在模式识别、计算机图形、计算机辅助设计等领域有广泛的应用[1,2]。

平面点集 Voronoi 图的常用构造算法有三类:分治法、平面扫描法和增量算法[1,2]。由于增量算法不仅适用于静态点集,而且还适用于动态点集,因而受到重视。

Voronoi 图增量算法中的关键工作是最近邻的选择和搜索。已有算法大都采用随机穷举法,因而效率较低。本文首先介绍了 Voronoi 图的翼边数据结构表示方法,在增量算法时间复杂性分析的基础上,提出了应用桶技术选择生成子并提高最近邻搜索效率的方法,并阐述了对退化情形和误差的处理方法,最后对算法进行了实验研究。结果表明,该算法具有较高的稳定性且算法的平均时间复杂性基本上是线性的。

2 Voronoi 图基本增量算法

2.1 Voronoi 图的定义

设 $P=\{p_1,p_2,\cdots,p_n\}\subset R^2$是二维欧氏空间上的点集,$d(\cdot,\cdot)$为欧氏距离。称

$$V(p_i)=\{p\in R^2 \mid d(p,p_i)\leqslant d(p,p_j),\ i,j=1,2,\cdots,n,i\neq j\}$$

为Voronoi 区域。其中,由点集 P 生成的 Voronoi 图可定义为

$$V(P)=\bigcup_{i=1}^{n}\{V(p_i)\}$$

Voronoi 区域的边,称为 Voronoi 图的边,Voronoi 区域的顶点称为 Voronoi 图的顶点。

2.2 翼边数据结构

翼边数据结构适用于表示连通的平面图。首先将 Voronoi 图扩充为几何图(平面图)。使用足够大的闭曲线环绕 Voronoi 图顶点,无限边交在此曲线上,闭曲线被分割成若干曲线段,也称为 Voronoi 边,称此图为扩充的几何图。

令

$$G=(V,E)$$

其中,$V=(v_1,v_2,\cdots,v_{n_v})$,$E=(e_1,e_2,\cdots,e_{n_e})$。

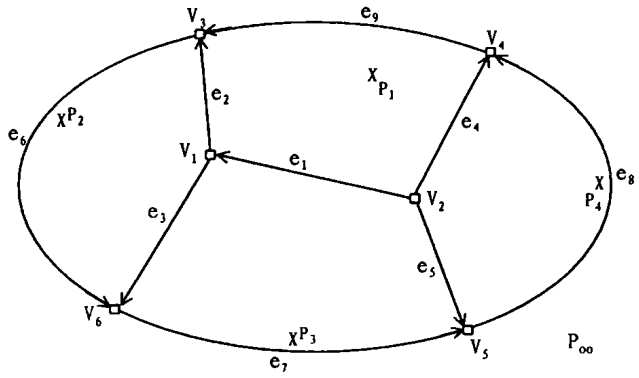

图1 翼边数据结构

首先对每一个边任选并固定一个方向,然后,给出顶点的序号$1,2,\cdots,n_v$和边的序号$1,2,\cdots,n_e$,并称生成子 p_i 的 *Voronoi* 多边形为多边形 $i(i=1,2,\cdots,n,n_\infty)$。翼边数据结构的基本内容如下所示:

[1]对多边形 $i(i=1,2,\cdots,n,\infty)$

(1)EdgeAroundPolygon[i]:多边形 i 边界上的一条边的序号。

[2]对顶点 j($j=1,2,\cdots,n_v$)

(1)EdgeAroundVertex[J]:与顶点相关联的一条边的序号。

(2)VertexW[j]:该顶点的属性,1表示普通点,0表示无穷远点。

(3)VerteX[j],VertexY[j]:点的坐标。无穷远点是单位方向向量。

[3]对每条边 k($k=1,2,\cdots,n_e$)

(1)RightPolygon[k]:边 k 右侧多边形的序号。

(2)LeftPolygon[k]:边 k 左侧多边形的序号。

(3)StartVertex[k]:边 k 起点的顶点序号。

(4)EndVertex[k]:边 k 终点的顶点序号。

(5)CwPredecessor[k]:在边 k 起点处,顺时针方向

*)得到国家自然科学基金(69973019)、辽宁省自然科学基金(9910200203)和辽宁省教委高等学校科研基金(990321081)的资助。廖士中 博士,教授,主要研究领域包括空间推理、计算几何、基于内容的检索。王晓东 硕士生,讲师,主要研究领域包括计算几何,基于内容的检索。

维普资讯 http://www.cqvip.com

上一条边的序号。

(6)CcwPredecessor[k]：在边 k 起点处，逆时针方向上一条边的序号。

(7)CwSucessor[k]：在边 k 终点处，顺时针方向下一条边的序号。

(8)CcwSucessor[k]：在边 k 终点处，逆时针方向下一条边的序号。

翼边数据结构可用13个数组表示。

2.3 基本增量算法

对 $l=1,2,\cdots,n$，设 V_l 是点集 $P=\{p_1,p_2,\cdots,p_l\}$ 的Voronoi图，增量算法的工作过程是对每一个 l 将 V_{l-1} 转换为 V_l，基本增量算法的执行过程如图2所示。

输入：点集 P，l，V_{l-1}
输出：V_l 的翼边数据结构
过程：
[1. 最近邻搜索]
找出 p_l 所在的 Voronoi 区域
[2. 生长边]
设 p_l 与所在 Voronoi 区域生成子 p_i 的垂直平分线与 $V(p_i)$ 的边界交于 w_1 和 w_2 两点，并使得 p_l 在有向线段 w_1w_2 的左侧，则得到 $V(p_l)$ 的一条边，并从此进入邻接的 Voronoi 多边形，以同样方式，找出 p_l 与相邻的多边形的生成子的垂直平分线线段序列，直到回到起点 w_1。
[3. 修改翼边数据结构]
删除 V_{l-1} 在 $V(p_l)$ 中的构造，并修改翼边数据结构。

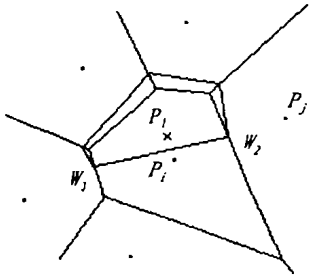

图2 构造 Voronoi 图的增量算法

3 增量算法的改进

3.1 时间复杂性分析

基本增量算法中，最近邻搜索可采用如下方法：设 p_i 是当前 Voronoi 图中的任一生成子，p_l 为待加入的生成子。若对于 p_i 周围的每一个生成子 p_j，都有

$$d(p_l,p_i)\leqslant d(p_l,p_j) \qquad (1)$$

成立，则 p_i 为最近邻生成子，否则找 p_i 周围离 p_l 最近的生成子 P_k，$k\Rightarrow i$，重复上述步骤直到(1)式成立。

该最近邻搜索算法的时间复杂性是 $O(\log l)$。

在基本增量算法中，生长边的时间复杂性是 $O(l)$，修改翼边数据结构的时间复杂性是 $O(l)$。因而，增量算法的(最坏)时间复杂性为 $O(n^2)$。

3.2 桶技术

由于 Voronoi 图的平均边数不会超过6，因而，若以均匀方式插入生成子，则增量算法生长边和修改翼边数据结构的时间复杂性都是 $O(1)$。进一步，若能以常数时间完成最近邻搜索，则可得到时间复杂性为 $O(n)$的增量算法。

桶技术可较好地完成上述工作[3]，其基本步骤如下所示。

[1]设所有生成子位于单位正方形 $S=\{(x,y)|0<x<1,0<y<1\}$ 中，把单位正方形均匀划分为 4^M 个单元(桶)，每个单元用二维坐标 $ADDRESS(I,J)$ 表示，见图3。其中，$M=\lfloor\log_4(\alpha^2 n)\rfloor$；$n$ 为生成子的个数；α^2 为每个生成子对应单元的数目，通常取 $\alpha^2=1$；$0\leqslant I\leqslant 2^M-1$，$0\leqslant J\leqslant 2^M-1$。

[2]构造一个深度为 M 的四叉树，并对叶子按从左到右顺序地从0到 4^M-1 编号，序号为 r 的叶子与坐标为 ADDRESS(I(r),J(r))的单元对应。这里 I(r),J(r)为有 2^M 个元素的数组，并按如下方式初始化：

```
c=[2^M/3], I(0)=c, J(0)=c, u=(-1)^(M+1), m=1, r=0, s=2c+u;
for t=1 to M do
{ for p=1 to m do
  { r=r+1, I(r)=s-I(r-m), J(r)=J(r-m)}
  m=2m
  for p=1 to m do
  {r=r+1, I(r)=I(r-m), J(r)=s-J(r-m)}
  m=2m, u=-2u, s=s+u
}
```

[3]按四叉树叶子的编号，从小到大依次地将叶子中的部分生成子加入到该叶子所有为空的祖先结点中。这样，可得到一棵树。除叶子外，每个结点至多有一个生成子(叶子可有多个生成子)，非空结点的父结点非空。

15	191	190	186	187	171	170	174	175	239	238	234	235	251	250	254	255
14	189	188	184	185	169	168	172	173	237	236	232	233	249	248	252	253
13	181	180	176	177	161	160	164	165	229	228	224	225	241	240	244	245
12	183	182	178	179	163	162	166	167	231	230	226	227	243	242	246	247
11	151	150	146	147	131	130	134	135	199	198	194	195	211	210	214	215
10	149	148	144	145	129	128	132	133	197	196	192	193	209	208	212	213
9	157	156	152	153	137	136	140	141	205	204	200	201	217	216	220	221
8	159	158	154	155	139	138	142	143	207	206	202	203	219	218	222	223
7	31	30	26	27	11	10	14	15	79	78	74	75	91	90	94	95
6	29	28	24	25	9	8	12	13	77	76	72	73	89	88	92	93
5	21	20	16	17	1	0	4	5	69	68	64	65	81	80	84	85
4	23	22	18	19	3	2	6	7	71	70	66	67	83	82	86	87
3	55	54	50	51	35	34	38	39	103	102	98	99	115	114	118	119
2	53	52	48	49	33	32	36	37	101	100	96	97	113	112	116	117
1	61	60	56	57	41	40	44	45	109	108	104	105	121	120	124	125
0	63	62	58	59	43	42	46	47	111	110	106	107	123	122	126	127
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

图3 四叉树增量算法单元序号图(M=4)

在按上述步骤构造树和排序生成子时，可使四叉树每一层的生成子尽可能均匀分布，即使得每个 Voronoi 区域的平均边数约为6。又由于每个待处理的生成子均在父结点生成子的邻域中，因此选择父结点内的生成子作为式(1)中的 p_i，可使最近邻搜索约在常数时间内完成。理想地，最近邻搜索的平均时间复杂性为 $O(1)$，生长边的平均时间复杂性为 $O(1)$，修改翼边数据结构的平均时间复杂性为 $O(1)$，此时增量算法的平均时间复杂性为 $O(n)$。

尽管采用上述方法构造 Voronoi 图时，平均时间复杂性可达 $O(n)$，但若输入的生成子位于阿基米德螺线 $\rho=a\phi$ 上，$0\leqslant\phi\leqslant 2\pi$，见图4，则最近邻搜索变得很复杂，即使采用桶技术也不能在常数时间内完成最近邻搜索，因而其时间复杂性仍为 $O(n^2)$。

3.3 基于桶技术的增量算法

基于桶技术的增量算法与基本增量算法的主要区别是，基于桶技术的增量算法需要对点集 P 中的生成子进行排序。

维普资讯 http://www.cqvip.com

算法过程如下。

输入：点集 P
输出：$V(P)$的翼边数据结构
过程：

[1. 初始化]
根据点集 P 中生成子的数量确定树的深度、单元(桶)的数量并对桶进行编号。

[2. 建立四元树]
构造一株深度为 M 的四叉树，对相应的结点进行初始化；
依次将点集 P 中的每个生成子加入到桶单元中；
将桶中的部分生成子加入到桶的空祖先结点中。

[3. 按序增量构造 Voronoi 图]
从树根开始，按先上层后下层、同一层次的先编号大后编号小的次序逐个加入生成子，增量地构造 Voronoi 图。

3.4 四点共圆处理

对于点 p_2和有向线段$\overrightarrow{p_0p_1}$，定义行列式

$$A(T)=\begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}$$

其中，p_i 的座标为(x_i, y_i)，$i=0,1,2$。若 $A(T)>0$，则点 p_2在有向线段$\overrightarrow{p_0p_1}$的左侧；若 $A(T)=0$，则点 p_2与有向线段$\overrightarrow{p_0p_1}$共线；若 $A(T)<0$，则点 p_2在有向线段$\overrightarrow{p_0p_1}$的右侧。

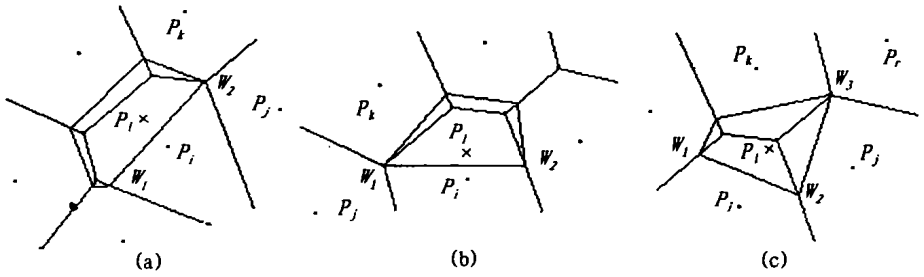

图5 共圆

在生长边的过程中，首先求出 p_l 与 p_i 的垂直平分线$\overline{w_1w_2}$，再计算 p_l 与 w_2所在边另一侧区域的生成子 P_j 的垂直平分线，依次逆时针方向求出 p_l 与其他相邻生成子的公共边，当回到 w_1点时结束。上述过程没有考虑四点共圆的情形。

图5(a)中，P_l 与 P_i 的垂直平分线$\overline{w_1w_2}$穿过 $V(p_i)$边界的顶点，此时 p_l, p_i, p_j, p_k 四点共圆，w_2为圆心。在按逆时针方向求第二条边时，由于 w_2为 $V(p_i)$的两个边的端点，这时，只需判断 p_l 与生成子 p_j, p_k 中的哪一个有公共边即可。图5(b)中，p_l, p_i, p_j, p_k 四点共圆，w_1为圆心，也存在类似问题。这两种情况的共同特点是与 p_l 有公共边的生成子 p_k 满足如下条件：(1)生成子 p_k 与 p_i 有公共边，(2)该公共边的某一端点为圆心，(3)该公共边在有向线段$\overrightarrow{w_1w_2}$的左侧。对图5(c)等情形，可转化为图5(a)或图5(b)的情形处理。

显然，该方法对多点共圆情形也是适用的。

在具体实现时，“共圆”的判定可根据具体的作图要求进行。若当前加入的生成子与已加入的一生成子的垂直平分线和某 Voronoi 边的交点到该 Voronoi 边一端点的距离 $d<\delta$，就可认为出现“共圆”，δ 可根据实际需要设置。

3.5 累积误差的处理

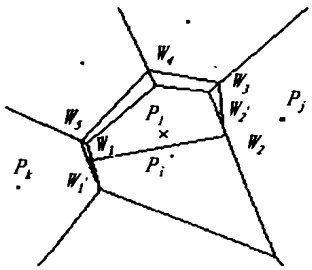

图6 误差的处理

在构造 Voronoi 图时，由于存在计算误差，计算出的两直线(线段)的交点不可能是精确的。如果计算次数较少，不会对 Voronoi 图的结构产生影响，但若是连续计算，则可能会产生累积误差，并可能产生错误的 Voronoi 图结构。如图6所示，在计算出 p_l 与 p_i 的垂直平分线段$\overline{w_1w_2}$后，在计算 p_l 与 p_j 的垂直平分线与 $V(p_j)$边界的交点 W_3时，若该垂直平分线用过点 w_2，斜率为 k(通过 p_l 与 p_j 求得)的点斜式方程来表示，则会产生累积误差。因而，以后计算得到的 w_4, w_5的坐标的误差就会越来越大，最终计算得到的 p_l 与 p_k 的垂直平分线与 $V(p_k)$边界的交点 w_1'与 w_1有较大的距离，甚至不在同一条边上，因而引起错误。为此，可仅用 p_l 与相应生成子的坐标来表示垂直平分线的方程，这样不会产生累积误差，从而构造出正确的 Voronoi 图。

4 实验结果

图7是本文算法构造的部分 Voronoi 图，其中，(a)为无共圆情形，(b)为有多点共圆和共线的情形。

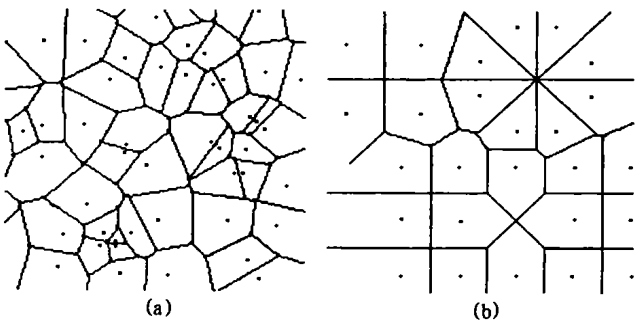

图7 Voronoi 图

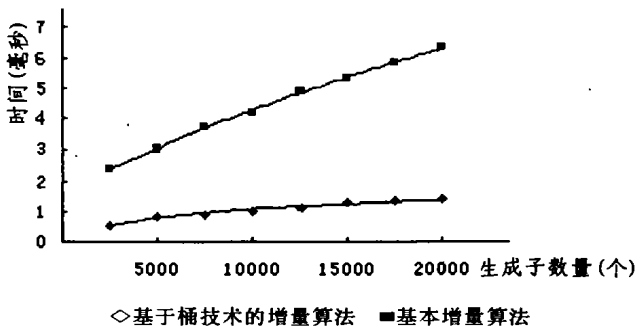

图8 采用不同算法构造 Voronoi 图时，平均每个生成子所用时间

(下转第53页)

维普资讯 http://www.cqvip.com

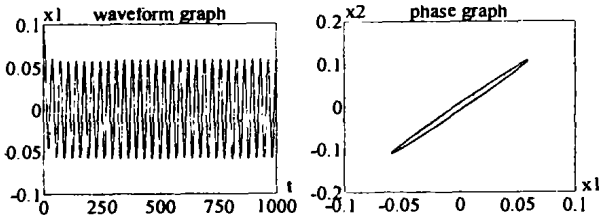

图 9 $a_1=0.5, a_2=1.8, b_1=0.8, b_2=0.3, \mu=0.14$，存在一个稳定的周期解。

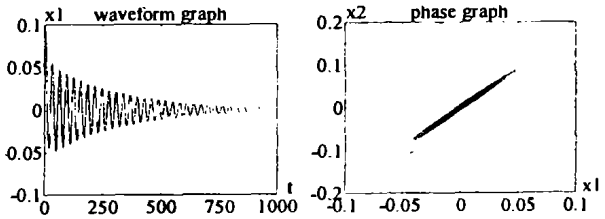

图 10 $a_1=0.5, a_2=1.8, b_1=0.8, b_2=0.3, \mu=0.17$，不存在周期解。

参 考 文 献

1 Hopfield J. Neurons with graded response have collective computational properties like those of two-state neurons. In: Proc. Nat. Acad. Sci. USA, 1984, 81: 3088～3092

2 Marcus C M, Westervelt R M. Stability of analog neural network with delay. Phys. Rev. 1989, A. 39: 347～359

3 Olien L, Belair J. Bifurcation, stability and monotonicity properties of a delayed neural model. Physica, 1997, D 102: 349～363

4 Belair J, Dufour S. Stability in a three-dimensional system of delay-differential equations. Can. Appl. Math. Quart., 1998, 4: 1878～1890

5 Gopalsamy K, Leung I. Delay-induced Periodicity in a neural network of excitation and inhibition. Physica, 1996, D 89: 395～426

6 Wei J, Ruan S. Stability and bifurcation in a neural network model with two delays. Physica, 1999, D130: 255～272

7 Gopalsamy K, Leung I. Convergence under dynamical thresholds with delays. IEEE Trans. on NN, 1997, 8(2): 341～348

8 Gopalsamy K, Leung I, Liu P. Global Hopf-bifurcation in a neural netlet. Appl. Math. Comput., 1998, 94: 171～192

9 Liao X F, Wu Z F, Yu J B. Hopf bifurcation analysis of a neural system with a continuously distributed delay. In: Proc. of the Intl. Symposium on Signal Processing and Intelligent System, Guangzhou, China, 1999

10 Liao X F, Wu Z F, Yu J B. Stability switches and bifurcation analysis of a neural network with continuous delay. IEEE Trans. SMC A, 1999, 29(6): 692～696

11 Liao X F, Wong K W, Wu Z F. Bifurcation analysis on a two-neuron system with distributed delays. Physica, 2001, D 149: 123～141

12 Liao X F, Wong K W, Leung C S, Wu Z F. Hopf bifurcation and chaos in a single delayed neuron equation with non-monotonic activation function. Chaos, Solitons and Fractals, 2001, 12: 1535～1547

13 Moiola J L, Chen G. Hopf Bifurcation Analysis: A Frequency Domain Approach. World Scientific, Singapore, 1996

14 Babcock K L, Westervelt R M. Dynamics of simple electronic neural networks with added inertia. Physica, 1986, D 23: 464～469

15 Babcock K L, Westervelt R M. Dynamics of simple electronic neural networks. Physica, 1987, D 28: 305～316

16 Destxhe A. Stability of periodic oscillation in a network of neurons with time delay. Phys. Lett., 1994, A 187: 309～316

（上接第 75 页）

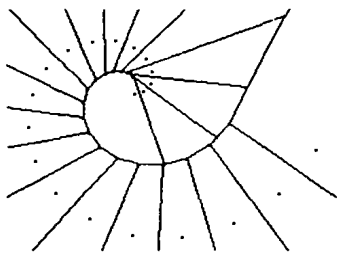

图 4 阿基米得螺线

表 1 算法效率的比较

生成子数量		2500	5000	7500	10000	12500	15000	17500	20000
基本增量算法	时间（秒）	6	15	28	42	61	79	101	126
	查找次数	83268	230486	421649	636747	891389	1143427	1495342	1826370
基于桶技术的增量算法	时间（秒）	1.3	4	6.5	10	14	19	23	28
	查找次数	4613	8364	13198	18518	24246	29896	29035	33457

表 1 列出了本文基于桶技术的增量算法与基本增量算法在执行时间和最近邻搜索次数的比较，这里 $\alpha=1$，生成子是随机生成的。实验环境为 Pentium 166/32M/1.2G，Windows 98/Visual C++5.0。图 8 是两种算法对每个生成子平均处理时间的比较。

结论 实验结果表明，本文给出的算法可处理平面上的任意点集，对三点共线、四点共圆等退化情形和误差的处理均简单有效。从表 1 可看出，采用桶技术对生成子排序后，最近邻搜索过程对每个生成子的平均查找次数不超过 2，即最近邻搜索过程大致可在常数时间内完成；而基本增量算法执行时，平均查找次数随生成子数目的增加而增加。在构造 Voronoi 图的时间上，两者也有明显的不同。基于桶技术的增量算法所用的时间明显小于基本增量算法所用的时间。图 8 清晰地表明，基于桶技术的增量算法在效率上有明显的改进，且每个生成子平均所用的 CPU 时间趋于常数，即本文给出的 Voronoi 图增量算法的平均时间复杂性基本上是线性的。

参 考 文 献

1 [美]F. P. 普雷帕拉塔, M. I. 沙莫斯. 计算几何导论[M]. 庄心谷译. 北京：科学出版社, 1990. 226～277

2 Ohya T, Iri M, Murota K. A Fast Vorovoi Diagram Algorithm with Quaternary Tree Bucketing[J]. Information Processing Letters, 1984, 18(4): 227～231

3 周培德. 计算几何－算法分析与设计[M]. 北京：清华大学出版社, 2000. 88～132