Journal of the Operations Research
Society of Japan
Vol. 27, No. 4, December 1984

IMPROVEMENTS OF THE INCREMENTAL METHOD FOR THE VORONOI DIAGRAM WITH COMPUTATIONAL COMPARISON OF VARIOUS ALGORITHMS

Takao Ohya
Central Research Institute of Electric Power Industry

Masao Iri
University of Tokyo

Kazuo Murota
University of Tsukuba

(Received May 13, 1983: Revised July 18, 1984)

Abstract A fast algorithm for the Voronoi diagram is proposed along with the performance evaluation by extensive computational experiments. It is shown that the proposed algorithm runs in linear time on the average. The algorithm is of incremental type, which modifies the diagram step by step by adding points (generators) one by one. What is new is a special preprocessing procedure for determining the order in which the generators are to be added, where we make use of a quaternary tree combined with an elaborate technique of "bucketing".

1. Introduction

For a set of n points P_i $(i=1,\ldots,n)$ in the Euclidean plane $\mathbf{R}^2$, the polygonal region defined by

$$(1.1) \qquad V_n(P_i) = \bigcap_{j \neq i} \{P \in \mathbf{R}^2 \mid d(P,P_i) < d(P,P_j)\}$$

is called the Voronoi region (or polygon), where $d(.,.)$ denotes the Euclidean distance. The planar skeleton V_n formed by the boundaries of $V_n(P_i)$ $(i=1,\ldots,n)$ is called the Voronoi diagram (sometimes called also the Dirichlet tessellation or the Thiessen tessellation), which plays a fundamental role in computational geometry [16] and finds applications in various fields such as geography, urban planning, ecology, physics and numerical analysis [7], [8], [15]. Vertices (edges) of the Voronoi diagram are called Voronoi points (Voronoi edges). We shall call P_i $(i=1,\ldots,n)$ the generators (or generating points) of the diagram. Two generators P_i and P_j are called contiguous in

V_n if their Voronoi regions $V_n(P_i)$ and $V_n(P_j)$ have a boundary edge in common.

In spite of its importance in practical applications, the Voronoi diagram had long been considered to be difficult to construct, especially when the number of generators is large, until the advent of computational geometry [15].

The divide-and-conquer algorithm, stated in [16], constructs the Voronoi diagram in $O(n \log n)$ time, which is known to be optimal (in the sense of the order of magnitude) with respect to the worst-case performance. This paper gives a complete description, as well as performance evaluation, of the quaternary incremental algorithm, which is one of the practical algorithms proposed and evaluated in [11], [12], [13], [14]. It is a variant of the seemingly primitive incremental method [2] which modifies the diagram step by step by adding generators one by one. What is new is a special preprocessing procedure for determining the order in which the generators are to be added, where we make use of an elaborate technique of "bucketing". Our use of buckets is only for ordering the generators in the incremental process, and is basically different from the use of buckets in [1]. Systematic experiments show that the proposed algorithm constructs the Voronoi diagram in $O(n)$ time on the average when n generators are distributed uniformly in the unit square, and that it is robust against the nonuniformity of the distribution of the generators.

2. Incremental Method

The basic idea [2] of the incremental method is given in this section. Though this algorithm has obviously the worst-case complexity $O(n^2)$, it runs in about $O(n^{3/2})$ time [2]. Furthermore, it can be polished up to run in $O(n)$ time in the sense of the average-case performance, which is the main objective of the present paper.

Suppose that n generators are ordered in some way or other from P_1 through P_n, and let V_m denote the Voronoi diagram for the first m generators $P_1, \dots, P_m$. Starting from a trivial diagram, say V_3, the incremental method constructs V_n through repeated modification of V_{m-1} to V_m $(m \le n)$, i.e., by adding a new generator P_m to the current diagram V_{m-1}. The m-th stage, i.e., the modification of V_{m-1} to V_m by adding P_m to V_{m-1}, consists of the following two phases.

Phase 1 (Nearest neighbor search): Among the generators $P_1, \dots, P_{m-1}$ of the diagram V_{m-1}, find the nearest, say $P_{N(m)}$, to P_m.

Phase 2 (Local modification): Starting with the perpendicular bisector of line segment $P_m P_{N(m)}$, find the point of intersection of the bisector with a boundary edge of $V_{m-1}(P_{N(m)})$ and determine the neighboring region $V_{m-1}(P_{N_1(m)})$ which lies on the other side of the edge; then draw the perpendicular bisector of $P_m P_{N_1(m)}$ and find its intersection with a boundary edge of $V_{m-1}(P_{N_1(m)})$ together with the neighboring region $V_{m-1}(P_{N_2(m)})$; ... ; repeating around in this way, create the region of P_m to obtain V_m, as illustrated in Fig 1. (See also [2].)

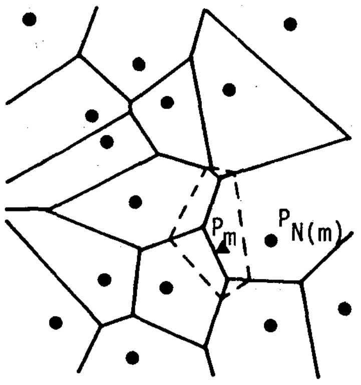


Fig. 1. Local modification of the Voronoi diagram in Phase 2 of the incremental method ($P_{N(m)}$ is the nearest neighbor of P_m.)

Note that some of the Voronoi edges of V_{m-1} do not remain in V_m, as is observed in Fig. 1.

Our task is to order the generators in such a way that, at each stage, Phase 2 may be done in constant time on the average and that it may be possible to find the nearest neighbor in Phase 1 in constant time on the average, too.

Phase 1 is equivalent to finding such $N(m)$ $(1 \le N(m) \le m-1)$ that $V_{m-1}(P_{N(m)})$ contains P_m. This problem is a kind of "point location" problem, but existing algorithms [8], [10], [16] for the point location problem cannot be directly applied, since the diagram varies from stage to stage. The following simple algorithm will be advantageous from the practical point of view.

Algorithm L:

Start with an initial guess $P_{i(0)}$ $(1 \le i(0) \le m-1)$ and set $i := i(0)$. If

(2.1) $$d(P_m, P_i) \le d(P_m, P_j)$$

for each P_j contiguous to P_i in V_{m-1}, then finish with $N(m) = i$ (in this case, it is proved without difficulty that P_i is the nearest to P_m [8]);

Otherwise set $i:=j$ for any j such that (2.1) fails and repeat this process.

Though this algorithm is of worst-case complexity $O(m)$, the initial guess, not specified above, is of crucial importance to its actual running time, and if the search area is restricted to a sufficiently small neighborhood of P_m, this algorithm can be expected to run in constant time, irrespectively of the number $m-1$ of the candidate generators.

As for Phase 2, the situation is rather subtle. Since the number of Voronoi edges meeting at a Voronoi point is not less than three, it follows from Euler's formula that the total number of Voronoi points of V_m is bounded by $2m$ and that of Voronoi edges of V_m by $3m$. (In case the generators are distributed randomly, these upper bounds are asymptotically tight.) As a consequence, the average number of edges of a Voronoi polygon is approximately equal to six. This alone does not guarantee, however, that the new region $V_m(P_m)$ created at the m-th stage with respect to a particular ordering of generators has a bounded number of edges independently of m, nor does it guarantee that Phase 2 can be performed in constant time. (See the experimental results for the consecutive cell algorithms described in the next section.) Nevertheless, it would intuitively be obvious and will not be very difficult to prove in more mathematical languages that, if the generators $P_1, \ldots, P_{m-1}$ are distributed approximately uniformly around P_m, the number of edges of $V_m(P_m)$ is nearly constant around six, so that the modification of the diagram in Phase 2 at each stage can be performed in constant time on the average. See Appendix 1.

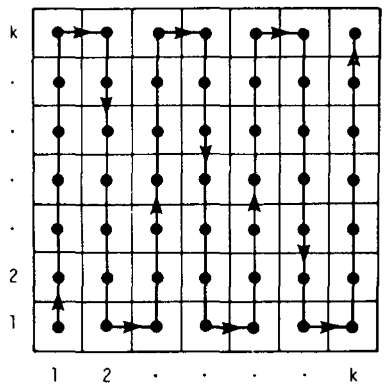
3. Orderings of Generators

From now on, the generators are assumed to lie in the unit square $S\equiv\{(x,y)\,|\,0<x<1, 0<y<1\}$. We partition S into k^2 square cells (or buckets) by dividing each side of the unit square into k equal parts, where $k=O(n^{1/2})$. Each generator belongs to one of those k^2 cells, and to which it belongs may be determined by multiplying its coordinates by k and then truncating the fractional parts off.

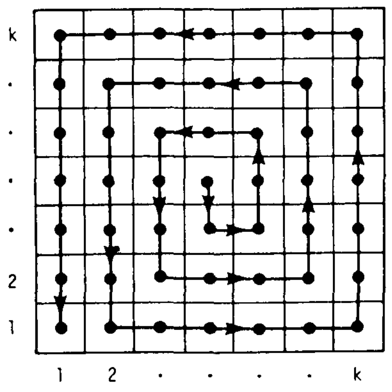
3.1. Consecutive cell algorithms

The first class of algorithms we have considered [14] is based on a consecutive cell order technique, which is similar to the one used in [5], [6] for the minimum-weight Euclidean matching problem. The value of the parameter $\alpha = kn^{-1/2}$ for the cell size is to be determined later. The cells are given a prescribed order, e.g., the "serpentine cell order" in Fig. 2(i), the "outward spiral cell order" in Fig. 2(ii), or the "inward spiral cell order". The n generators are ordered to form a sequence $P_1, \ldots, P_n$, which is consistent with the cell order, i.e., the generators belonging to one and the same cell may be ordered arbitrarily among themselves. Then the incremental method above is applied to the generators in this order; in Phase 1 at the m-th stage, we take the preceding generator P_{m-1} as the initial guess for the nearest neighbor of P_m, that is, we set $i(0)=m-1$ in Algorithm L.

Regrettably, as the experiments show, this class of algorithms even with the tuned parameter value of α, runs in time slightly longer than $O(n)$ on the average. (This is, however, considerably better than the divide-and-conquer type algorithm; see Section 4 for detail.) Closer investigation revealed that the computation needed in Phase 1 as well as that in Phase 2 required nonlinear time. In fact, in Phase 1, the number of generators which must be visited before the nearest neighbor of P_m is found increases with m, and, in Phase 2, the number of edges of $V_m(P_m)$ grows unboundedly, though very slowly, with m. (This does not contradict the fact that the number of Voronoi edges of the final diagram V_n is bounded by $3n$, since part of the Voronoi edges of $V_m(P_m)$ do not remain in the final diagram but disappear in the course of subsequent computation.)



(i) Serpentine cell order



(ii) Outward spiral cell order

Fig. 2. Typical cell orders

3.2. Quaternary incremental algorithm

The second improved algorithm, named the quaternary incremental algorithm, is as follows.

We take $k=2^M$, where $M=\max(\lfloor \log_4(\alpha^2 n)\rfloor,1)$ with parameter α to be properly chosen (see below), where we have $\alpha n^{1/2} \leq k < 2\alpha n^{1/2}$. The generators are ordered by means of a quaternary tree T of depth M; T has 4^M $(=k^2)$ leaves corresponding to the k^2 cells, and a node at depth h represents a square of side 2^{-h}. In particular, the root, i.e., the node at depth 0, represents the entire unit square.

We number the leaves of T from 0 to 4^M-1 in the natural manner from left to right; four consecutive nodes grouped in fours from an end at the same depth has a common father. Each cell is given a two-dimensional address (I,J) $(0 \leq I \leq k-1,\ 0 \leq J \leq k-1)$ in the natural way. The leaf r is identified with the cell $(I(r),\ J(r))$ under the correspondence defined by the following procedure, of which the computational complexity is $O(n)$:

```
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 .
```

An example of this cell order is shown in Fig. 3 for $M=4$ $(k^2=256)$.

As is easily seen intuitively from Fig. 3 and is confirmed from the procedure shown above, the numbering starts from a cell located approximately at a point of the unit square with coordinates (1/3, 1/3), and is extended to the whole square by repeated reflections with respect to a vertical and a horizontal line, alternately.

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
J/I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 3. Numbering of cells of the quaternary incremental algorithm (M=4)

7	[47] 24	[46] 59	[42] 60	[43] 25	[59]	[58]	[62] 61 14	[63] 37
6	[45]	[44] 64 16	[40] 57	[41] 52	[57]	[56]	[60] 15 5	[61]
5	[37] 23	[36]	[32] 17	[33] 18 7	[49] 33 8	[48] 29 9 1	[52] 36 32	[53] 53 51
4	[39]	[38] 22 13	[34]	[35] 19	[51] 31	[50] 54	[54]	[55] 50 49
3	[7] 45 43 3	[6]	[2] 30 27	[3] 2	[19] 63	[18] 62	[22] 20 4	[23]
2	[5] 46	[4]	[0] 11	[1] 10	[17] 55	[16] 58 28	[20]	[21] 48
1	[13] 6	[12] 26	[8] 44 38	[9] 21 12	[25]	[24] 39	[28] 35	[29] 65
0	[15] 47	[14]	[10] 34	[11] 66	[27] 42 41	[26]	[30]	[31] 56 40
J/I	0	1	2	3	4	5	6	7

Fig. 4. An example of distribution of 66 generators in 64 cells (M=3, k=8)

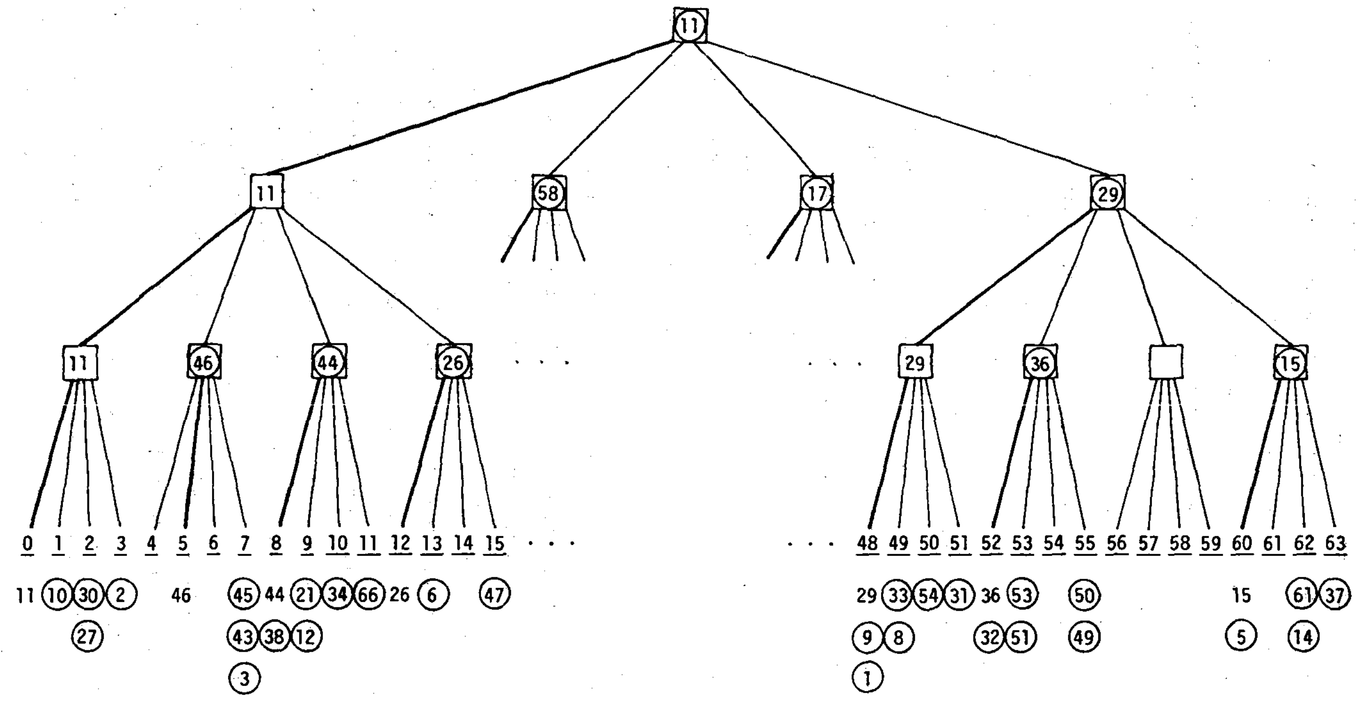


Fig. 5. The quaternary tree *T* for the example in Fig. 4

The leaves of T, now being identified with the cells, have the generators in the respective cells, whereas the other nodes of T are still empty. Then, scanning the leaves from left to right, i.e., from $r=0$ to $r=k^2-1=4^M-1$, we pick up a single generator, if any, contained in a leaf and put it in all the empty ancestor nodes of the leaf. Thus we obtain a quaternary tree, whose nodes, except for leaves, are filled with at most one generator. (A leaf may have more than one generator.) Note that a nonempty node has a nonempty father.

The following example will illustrate the construction of the quaternary tree T, where we set $\alpha=1$. Suppose that $n=66$ generators are given as in Fig. 4, where the generators are indicated simply by their numbers and the cell numbers of the 64 cells are shown in brackets [ ]. Then, the corresponding quaternary tree T of Fig. 5 is of depth $M=3$, having $4^M=64$ leaves, each of which is identified with one of the 64 cells. The first (left-most) leaf, leaf 0, corresponding to the cell $(I,J)=(2,2)$, contains a generator P_{11}, which is put in all the ancestor nodes up to the root. For the generator P_{10} contained in the next leaf, leaf 1, nothing is done, since its father node is already filled with P_{11}. Going on towards the right in this way, we may hit upon an empty leaf like leaf 4. Then we do nothing. After scanning all the 64 leaves, we have the quaternary tree T of Fig. 5. Note that an intermediate node, i.e., the father of leaves 56 to 59, is still empty, since none of its descendent leaves contain a generator.

The ordering of the generators in the incremental process is determined as follows. The tree T is traversed from the root in the breadth-first manner, from right to left at the same depth. Every time we encounter a node with a "new" generator, we add it. Since the father of that node must contain a generator which has already been processed, that generator is adopted as the initial guess $P_{i(0)}$ of the nearest neighbor search in Phase 1. When a leaf contains more than one generator left unadded, they are added in an arbitrary order among themselves, where the generator in the same cell which was added previously may be adopted as the initial guess $P_{i(0)}$ in Phase 1.

In the tree T of Fig. 5, the circles indicate the "new" generators to be added to the Voronoi diagram during the traversal of T. In the present case, the generators are added in the following order: P_{11}, P_{29}, P_{17}, P_{58}; P_{15}, P_{36}, ..., P_{26}, P_{44}, P_{46}; P_{37}, P_{61}, P_{14}, P_5, ..., P_9, P_1, ..., P_{47}, ..., P_{30}, P_{27}, P_{10}. As for the initial guess of the nearest neighbor, we use, e.g., P_{29} for P_{15} and P_{36}; P_{11} for P_{26}, P_{44} and P_{46}; P_{15} for P_{37} and P_{61}; P_{61} for P_{14}; P_{29} for P_9; and P_9 for P_1.

4. Experimental Results

4.1. Performance of several algorithms for uniformly distributed generators

Throughout the experiments, we made use of the following "full" data structure for representing generators and Voronoi diagrams, where n is the total number of generators given:

(i) coordinates of generators ($2n$ words);

(ii) coordinates of Voronoi points ($4n$ words);

(iii) Voronoi points incident to Voronoi edges ($6n$ words);

(iv) generators defining Voronoi edges ($6n$ words);

(v) incidence list of Voronoi edges to Voronoi points and to Voronoi regions ($15n$ words).

Recall that the total number of Voronoi points is bounded by $2n$ and that of Voronoi edges by $3n$. Our data structure requires $33n$ words in total for n generators. The "compact" data structure, of which the idea is found in [3], requires $14n$ words, consisting of (i), (iv) and part of (v), i.e.,

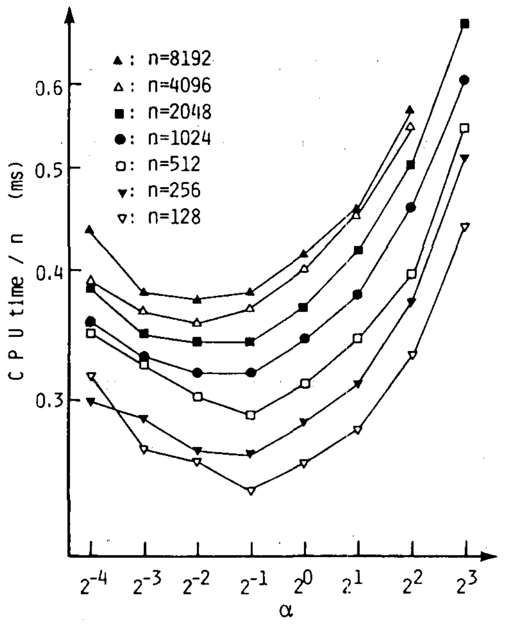
(v') the edges which are clockwise adjacent to each edge at its both ends ($6n$ words).

(The latter is more efficient in space, but the computational performance is less efficient in time, i.e., the computation time needed for the latter is about 2.5 times as much as that for the former, according to our computational experience.) The experimental computations were made on HITAC M-280H (VOS3/JSS4, Optimizing FORTRAN77, OPT=2) at the Computer Centre of the University of Tokyo with double precision arithmetic with mantissa of 14 hexadecimal digits.

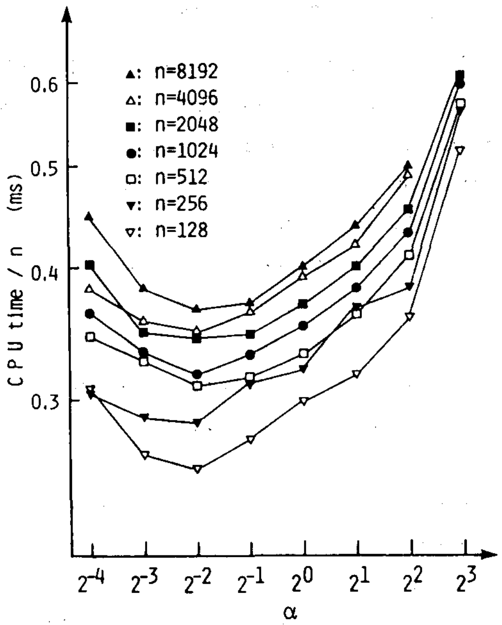
The CPU times needed for constructing the Voronoi diagram by several algorithms were measured mainly in the case where generators are distributed uniformly in the unit square. The results are shown by the scale of "CPU time /n" (the time per generator!), for the average of 10 independent instances, for n from 2^7 (=128) to 2^{13} (=8192) or to 2^{15} (=32768).

In the first place, the optimal values of the parameter α of the consecutive cell algorithms are determined empirically. Fig. 6 suggests that the optimal values lie around 0.25 for any of the three cell orders considered here, when the number n of generators is sufficiently large.

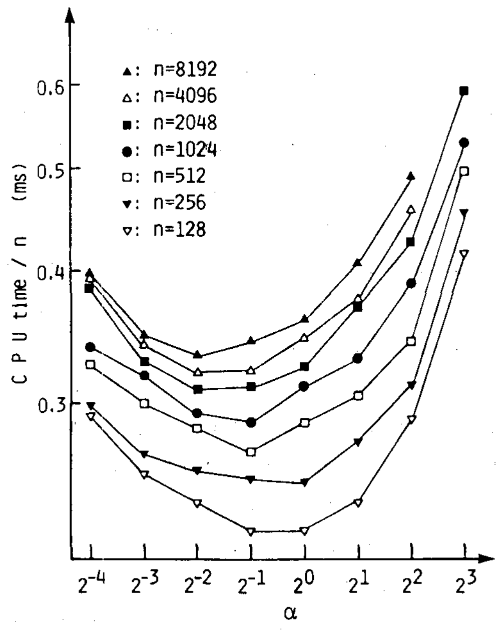
The consecutive cell algorithms with different cell orders are compared in Fig. 7. All of them worked fairly well and their performances seem practically satisfactory. It is surprising to see that the incremental method with such a simple preprocessing is very effective compared with the primitive incremental method where the generators are added in a random order without any particular preprocessing; the latter takes about $O(n^{3/2})$ time on the



(i) Serpentine cell order



(ii) Inward spiral cell order



(iii) Outward spiral cell order

Fig. 6. The experimental search for the optimal value of the parameter α of the consecutive cell algorithms

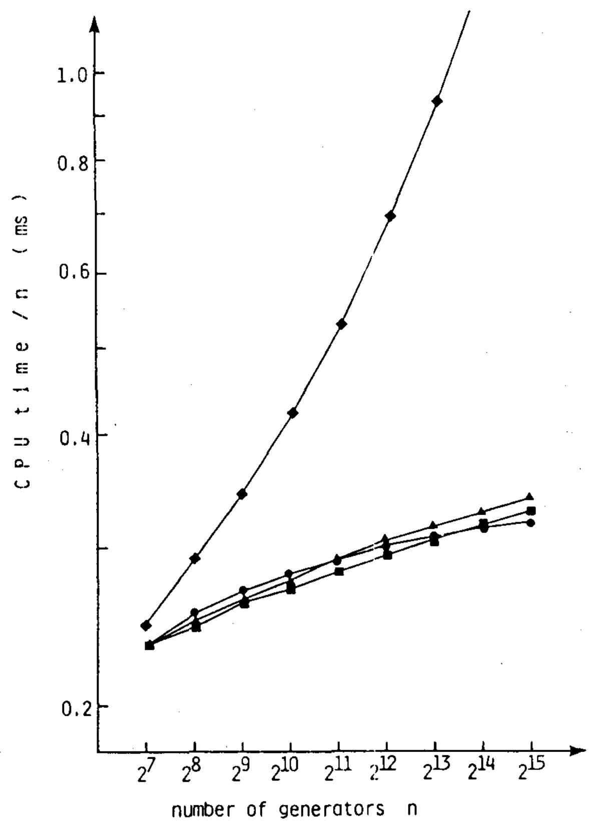


Fig. 7. CPU time per generator for constructing the Voronoi diagram of uniformly distributed generators by consecutive cell algorithms with different cell orders (α=0.25)

◆ : primitive incremental method
▲ : incremental method with serpentine cell order
● : incremental method with inward spiral cell order
■ : incremental method with outward spiral cell order

average. In spite of the drastic improvement in performance due to the preprocessing, the consecutive cell algorithm did not run in $O(n)$ time.

Next, the optimal value of the parameter α of the quaternary incremental algorithm was determined again by experiment. On the basis of the observation that the average running time of the quaternary incremental algorithm per generator is independent of the total number n of generators (see Fig. 9), the optimal value of parameter α was determined to be approximately equal to unity from the performance for n=8192 generators shown in Fig. 8. (Since the computation time of the quaternary incremental algorithm is nearly proportional to n for $n \geq 1024$, it suffices to investigate the effect of the value of parameter α for a single n.) It was also observed that the performance is not very sensitive to the variation of the parameter value. With the choice of parameter value $\alpha=1$, the asymptotically constant time of 0.26ms was needed per generator, independently of the total number n of the generators involved (see also Fig. 9).

Finally, the performance of the several algorithms are compared in Fig. 9 when the generators are distributed uniformly in the unit square. The program we wrote according to the divide-and-conquer algorithm [16] of worst-case complexity $O(n \log n)$, with either "full" or "compact" data structure, did not run in $O(n)$ time even on the average (see Fig. 9). The code with the "full" data structure ran about twice as fast as that with the "compact" structure. The consecutive cell algorithm with, say, the outward spiral cell order is substantially faster than the divide-and-conquer algorithm. The quaternary incremental algorithm was found to be quite efficient.

We have not tested the variants of incremental algorithms with the compact data structure, but the effect of the difference in data structures will be the same for the incremental algorithms as for the divide-and-conquer algorithms.

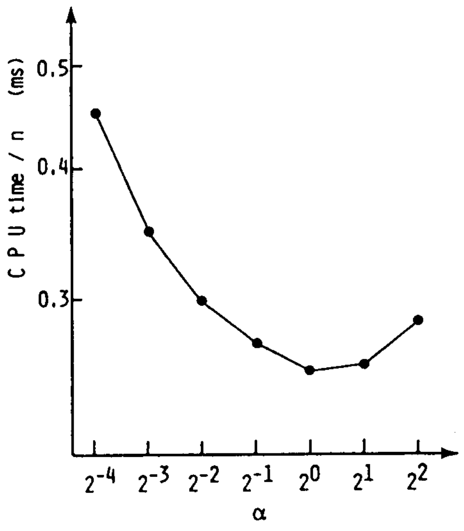


Fig. 8. The experimental search for the optimal value of the parameter α of the quaternary incremental algorithm

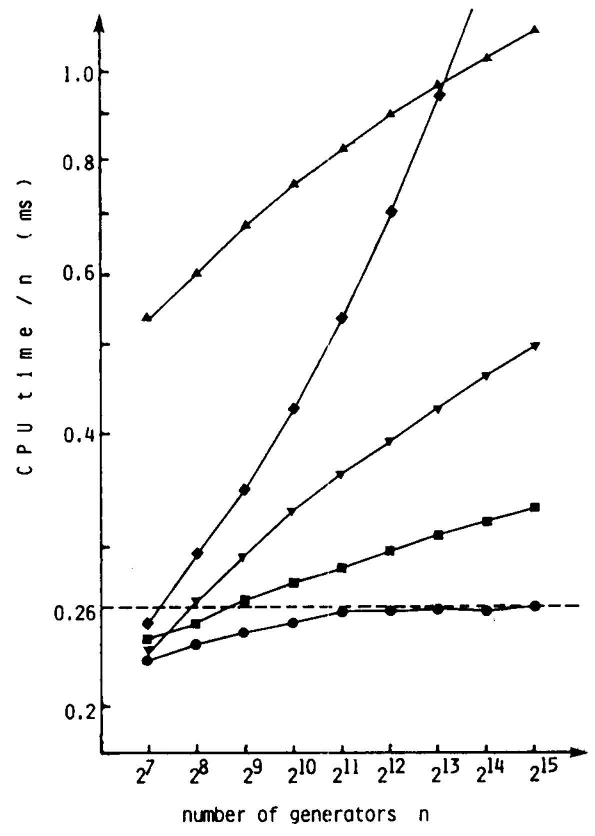


Fig. 9. CPU time per generator for constructing the Voronoi diagram of uniformly distributed generators by various algorithms

▲ : divide-and-conquer algorithm with "compact" data structure
▼ : divide-and-conquer algorithm with "full" data structure
◆ : primitive incremental method
■ : incremental method with outward spiral cell order (α=0.25)
● : quaternary incremental algorithm (α=1)

4.2. Robustness of the quaternary incremental method against nonuniformity of the distribution of generators

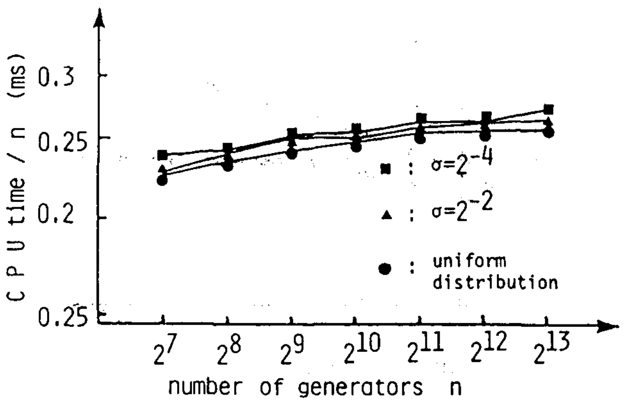
In this subsection, we will investigate the sensitivity of the quaternary incremental algorithm to the distribution of generators, i.e., how the performance of the algorithm changes when the distribution of generators deviates from the uniform. As typical nonuniform distributions, we have considered the four cases below.

[Case 1] Uncorrelated bivariate normal distribution

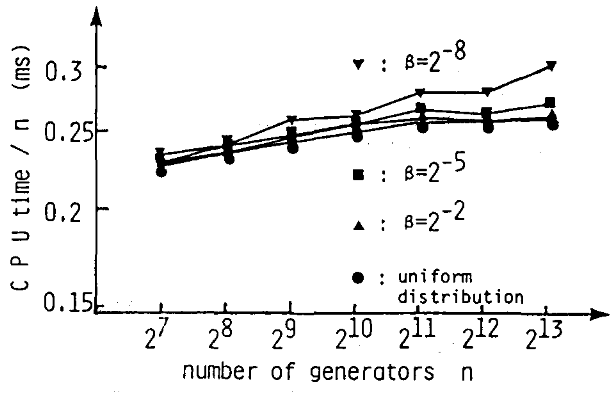
The generators are distributed subject to the normal distribution:

$$N\left(\begin{pmatrix}1/2\\1/2\end{pmatrix},\ \sigma^2\begin{pmatrix}1&0\\0&1\end{pmatrix}\right),$$

where the generators lying outside the unit square S are ignored. An instance of this distribution with n=128 and $\sigma=2^{-2}$ is shown in Fig. 11(i). The CPU times per generator required by the quaternary incremental algorithm are shown in Fig. 10(i). There seems to be no big difference from those for the uniform distribution and the total CPU time for n generators is approximately proportional to n.

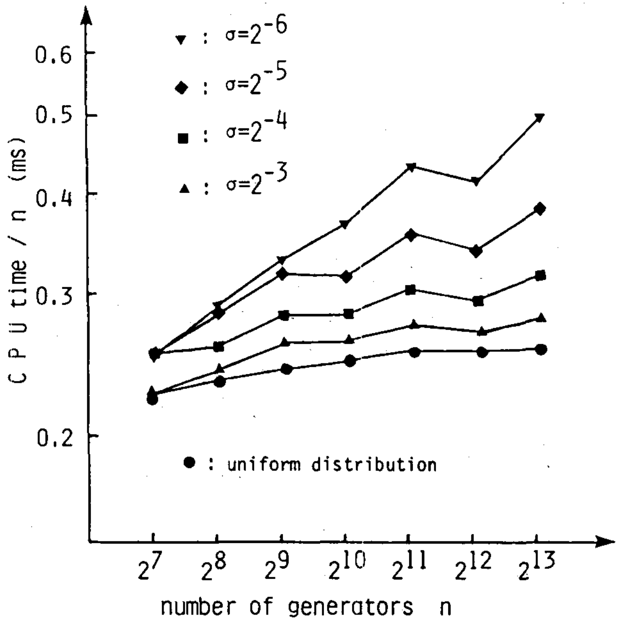


(i) Case 1: Uncorrelated normal distribution

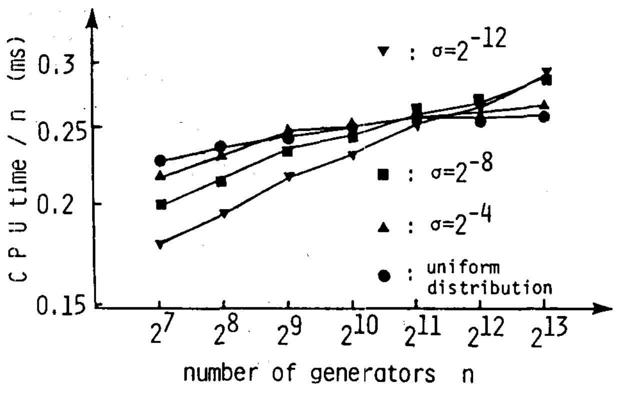


(ii) Case 2: Distribution concentrated along a line

Fig. 10. CPU time per generator for constructing the Voronoi diagram of nonuniformly distributed generators by the quaternary incremental algorithm (to be continued)

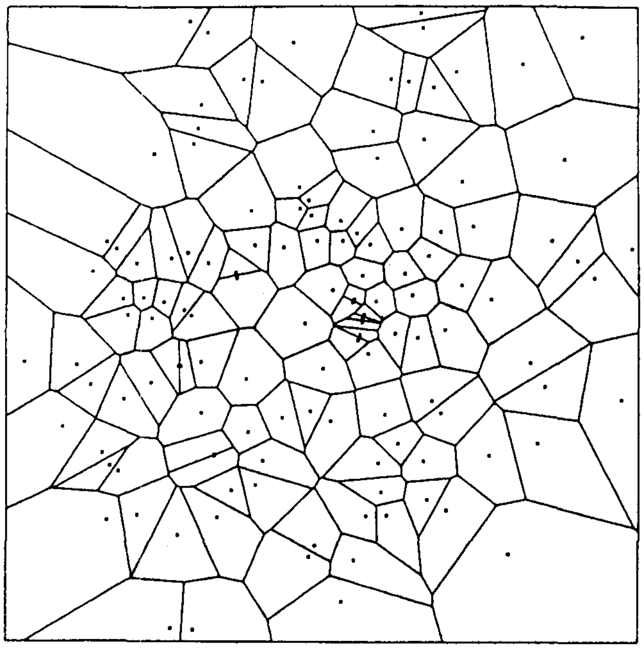


(iii) Case 3: Mixture of normal distributions

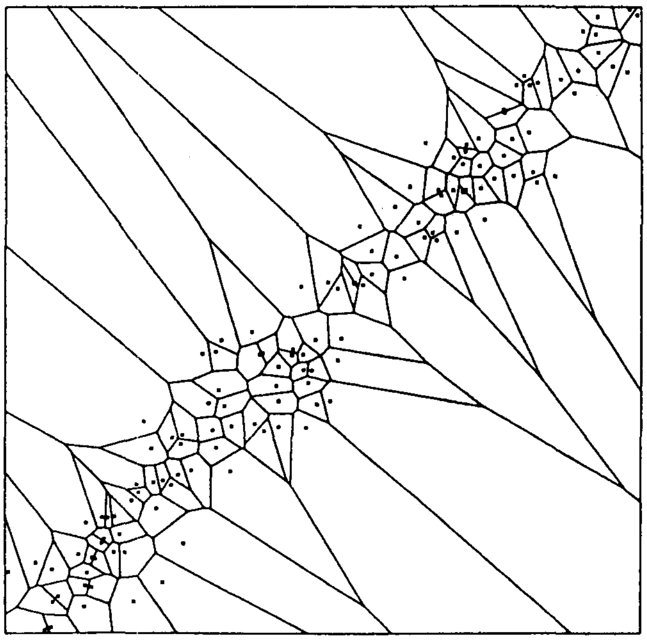


(iv) Case 4: Distribution concentrated along a circle

Fig. 10. CPU time per generator for constructing the Voronoi diagram of nonuniformly distributed generators by the quaternary incremental algorithm (continued)

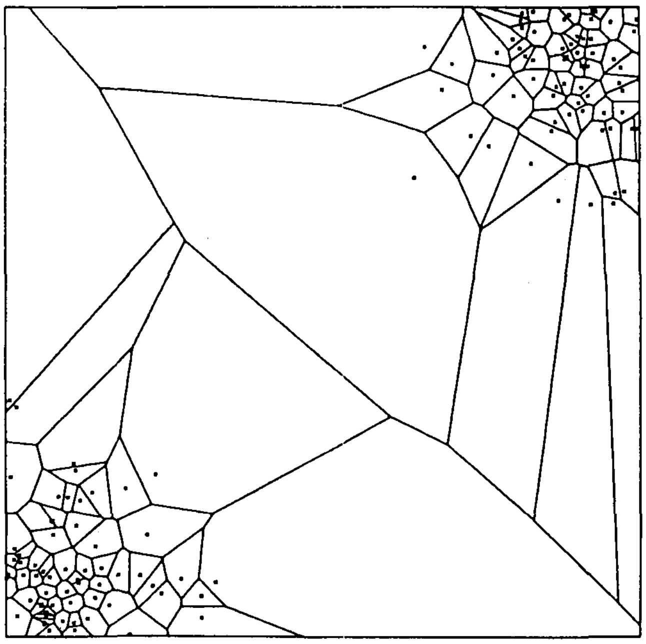


(i) Case 1: Uncorrelated normal distributions ($\sigma=2^{-2}$)



(ii) Case 2: Distribution concentrated along a line ($\beta=2^{-5}$)

Fig. 11. Examples of Voronoi diagrams of nonuniformly distributed generators (to be continued)


(iii) Case 3: Mixture of normal distributions ($\sigma=2^{-3}$)

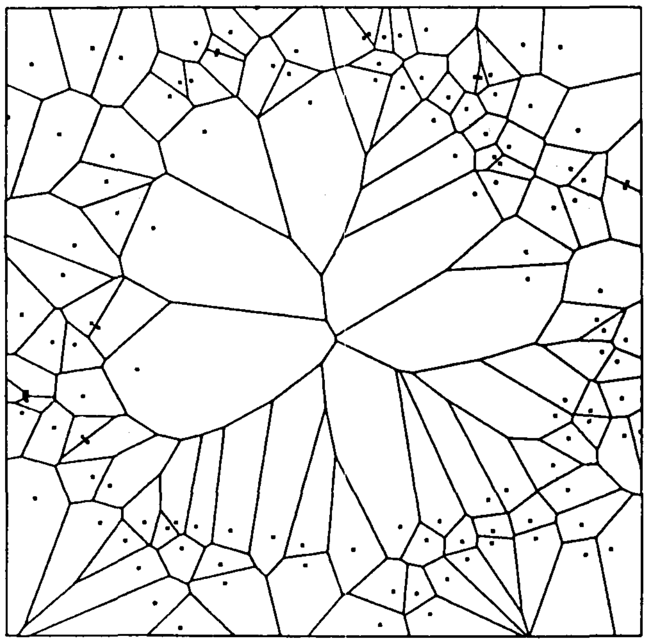

(iv) Case 4: Distribution concentrated along a circle ($\sigma=2^{-4}$)

Fig. 11. Examples of Voronoi diagrams of nonuniformly distributed generators (continued)

[Case 2] Distribution concentrated along a line.
The generators are distributed subject to a highly correlated normal distribution:

$$N\left(\begin{pmatrix}1/2\\1/2\end{pmatrix},\ \begin{pmatrix}1+\beta^2 & 1-\beta^2\\1-\beta^2 & 1+\beta^2\end{pmatrix}\right),$$

where the generators outside the unit square are ignored. For β small, the generators are clustered along the diagonal line, as is illustrated in Fig. 11 (ii) with n=128 and $\beta=2^{-5}$. The CPU times per generator required by the quaternary incremental algorithm are shown in Fig. 10(ii). Even in the extreme case of $\beta=2^{-8}$, the running time is about 1.2 times as much as that for the uniform distribution.

[Case 3] Mixture of normal distributions
Among the n generators in the unit square S, $n/2$ generators are taken from the normal distribution:

$$N\left(\begin{pmatrix}\sigma\\\sigma\end{pmatrix},\ \sigma^2\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}\right)$$

and the remaining $n/2$ from

$$N\left(\begin{pmatrix}1-\sigma\\1-\sigma\end{pmatrix},\ \sigma^2\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}\right).$$

As σ becomes smaller, the generators tend to cluster around the two diagonally opposite corners. The CPU times per generator by the quaternary incremental algorithm are shown in Fig. 10(iii). Though the performance of the quaternary incremental algorithm appears most sensitive to this type of nonuniformity of generators, it is still good enough for such distribution as is shown in Fig. 11(iii), where n=128 and $\sigma=2^{-3}$.

[Case 4] Distribution concentrated along a circle
Points are randomly generated in such a way that the distances of generators from the center (1/2, 1/2) are normally distributed subject to $N(1/2-\sigma,\ \sigma^2)$ and their angles around the center are subject to the uniform distribution. (Those outside the unit square are ignored.) An example is shown in Fig. 11 (iv), where n=128 and $\sigma=2^{-4}$. The CPU times per generator required by the quaternary incremental algorithm are shown in Fig. 10(iv). No serious deterioration of the performance is observed. It may seem strange that it takes less time than in the case of the uniform distribution when both n and σ are small, e.g., $n=2^7$ and $\sigma=2^{-12}$. This phenomenon may be explained as follows. If the generators are clustered along a circle, the number of generators lying

on the boundary of the convex hull of the generators is significantly larger than that in the case of uniform distribution, and consequently the total number of Voronoi edges is considerably smaller, especially when n is moderately small.

For comparison, the CPU times of the consecutive cell algorithm with the outward spiral cell order (α=0.25) are shown in Fig. 12 when the generators are distributed as in Case 3 above. Comparing Fig. 12 with Fig. 10(iii), it may be seen that the quaternary incremental algorithm is considerably more robust than the consecutive cell algorithms.

5. Conclusion

In spite of the worst-case complexity $O(n^2)$, the quaternary incremental algorithm constructs the Voronoi diagram for n generators in $O(n)$ time on the average, much faster than the divide-and-conquer algorithm of worst-case complexity $O(n \log n)$. It is said that algorithms of divide-and-conquer type sometimes fail for large problems, probably because generators are divided

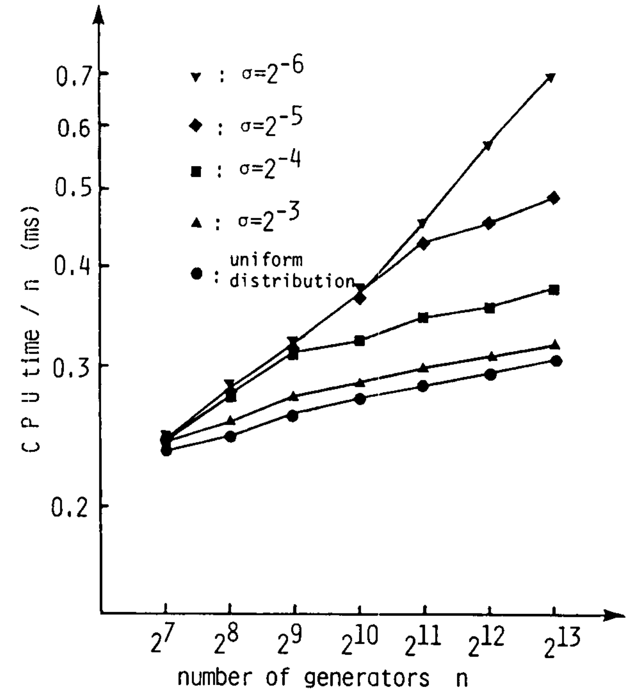


Fig. 12. CPU time per generator for constructing the Voronoi diagram of nonuniformly distributed generators (Case 3: Mixture of normal distributions) by the consecutive cell algorithm with outward spiral cell order (α=0.25)

into so thin strips that many almost parallel Voronoi edges are created and their intersections are to be determined in the course of computation. It is noteworthy, on the other hand, that the incremental method is free from this type of numerical difficulty. It is also verified by experiments that the quaternary incremental algorithm is robust against the nonuniformity of the generators. Considering the running time and the ease in coding, the "full" data structure is recommended, if space permits, rather than the "compact" one. The extension of the ideas presented in this paper for the incremental method to the case of more general Voronoi diagrams such as the Voronoi diagram of line segments and polygons [9], that in the Laguerre geometry [4], etc., should deserve due investigation.

Acknowledgement

The authors thank members of their research group, especially Mr. H. Imai, for their cooperation and discussions, and Professor D. Avis of McGill University, Montreal, and his research group on computational geometry for valuable information and comments.

Part of this research was supported by Grand in Aid of the Ministry of Education, Science and Culture of Japan.

Appendix 1. A Theoretical Analysis of the Average-Time Complexity of the Quaternary Incremental Algorithm

This appendix affords a theoretical support for the linearity of the average-case time complexity of the quaternary incremental algorithm, which has been confirmed experimentally in Section 4. For that purpose the following properties will play a fundamental role.

(1) The expected value of the number Y_m of the generators to be visited by Algorithm L in Phase 1 at stage m is bounded by a constant depending on α:

(A1.1) $$E[Y_m] \leq C_1.$$

(2) The expected value of the number X_{mi} of Voronoi edges of a Voronoi polygon $V_m(P_i)$ of any intermediate diagram V_m is bounded by another constant depending on α:

(A1.2) $$E[X_{mi}] \leq C_2.$$

From these properties we may expect that the average complexity of Phase 1 at each stage of the quaternary incremental algorithm is O(1), since the number of the generators P_i to be visited before $P_{N(m)}$ is found is O(1) by (A1.1) and since the amount of computation, for each P_i, needed to find a contiguous generator P_j that violates (2.1) is of the order of the number of edges of $V_{m-1}(P_i)$, which is also O(1) by (A1.2). Note also that (A1.2) guarantees, in particular, that the expected number of Voronoi edges created in Phase 2 at each stage is O(1) and therefore we may expect that Phase 2 can be done in O(1) time, since the amount of computation to determine $P_{N_{i+1}(m)}$ from $P_{N_i(m)}$ is of the order of the number of edges of $V_{m-1}(P_{N_i(m)})$, which is also O(1) by (A1.2).

In the following, we will briefly demonstrate how the bounds (A1.1) and (A1.2) are established, where we assume that P_m is added at a node of depth h of the quaternary tree and set $c=2^{-h}$ $(1 \le h \le M)$, which represents the size of the supercell corresponding to the node. Note that the relation

(A1.3) $$\alpha^2 n/4 < 4^M \le \alpha^2 n$$

holds in this case.

A1.1. Inequality (A1.1)

Let $P_{i(0)}, P_{i(1)}, \ldots, P_{i(Y_m)} = P_{N(m)}$ be the generators visited in Phase 1 at stage m. Since the distances to P_m from these generators decrease monotone, they must be contained in the disk $D(P_m,r)$ of radius $r=d(P_m,P_{i(0)})$ centered at P_m. Both P_m and $P_{i(0)}$ are contained in a square of side $2c$, so that

(A1.4) $$r \le 2\sqrt{2}\,c.$$

Obviously, Y_m is smaller than the number of generators contained in the disk $D(P_m,r)$, which is included in $D(P_m,2\sqrt{2}\,c)$ by (A1.4).

In case P_m is added at an intermediate node $(h \le M-1)$, each supercell of side c contains at most one generator already added, and therefore

(A1.5) $$Y_m \le 49,$$

since $D(P_m,2\sqrt{2}\,c) \cap S$ intersects at most $(\lceil 4\sqrt{2}\rceil+2)^2=49$ supercells of side c. When P_m is added at a leaf $(h=M)$, the expected number of generators, among n generators, which are contained in $D(P_m,2\sqrt{2}\,c) \cap S$, gives an upper bound on $E[Y_m]$:

(A1.6) $$E[Y_m] \le (n-1)\pi(2\sqrt{2}\,c)^2 \le 32\pi/\alpha^2.$$

From (A1.5) and (A1.6), the bound (A1.1) is obtained with

(A1.7) $$c_1 = \max(49,\ 32\pi/\alpha^2).$$

A1.2. Geometric Lemma

Consider a Voronoi diagram V for a set of generators including four noncollinear points Q_i $(i=0,\dots,3)$. The half plane determined by line Q_0Q_1 (resp. Q_0Q_2) and lying on the opposite side of Q_2 (resp. Q_1) is denoted by H_1 (resp. H_2). Denote by D the circumcircle (including the interior) of the triangle $Q_0Q_1Q_2$ (cf. Fig. A1.1).

Lemma A1.1. If Q_0 and Q_3 are contiguous in V, then $Q_3 \in H_1 \cup H_2 \cup D$.

Proof: The assertion follows from the fact that if Q_0 and Q_3 are contiguous in V, there exists a circle C such that both Q_0 and Q_3 are on its circumference and that it contains no generator in its interior. Q.E.D.

Let r_i be the length of line segment $Q_0Q_i (i=1,2)$ and β the angle $Q_1Q_0Q_2$. Suppose that a point P lies on the circumference of D interior to the angle $Q_1Q_0Q_2$ and let r_0 denote the distance of P from Q_0, i.e., $r_0=Q_0P$.

Lemma A1.2. If $\pi/4 \le \beta \le 3\pi/4$, then $r_0^2 \le 2(r_1^2+r_2^2+\sqrt{2}\,r_1r_2)$.

Proof: The diameter of D is greater than or equal to $r_0=Q_0P$:

(A1.8) $$r_0 \le (r_1^2+r_2^2-2r_1r_2\cos\beta)^{1/2}/\sin\beta.$$

For β such that $\pi/4 \le \beta \le 3\pi/4$, (A1.8) takes its maximum when $\beta=3\pi/4$. Q.E.D.

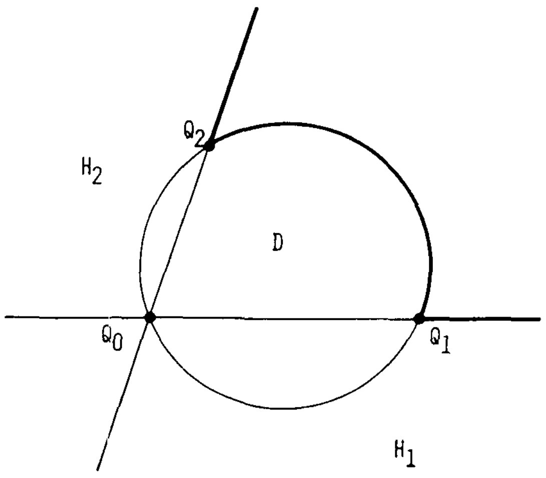


Fig. A1.1. Admissible region for a generator Q_3 contiguous to Q_0

A1.3. Inequality (A1.2)

For each P_i, we partition the unit square S into 8 parts $S^{(j)}$ $(i \le j \le 8)$, which we call sectors, by 8 rays emanating from P_i with the angle of $\pi/4$ between two consecutive ones, as illustrated in Fig. A1.2. If we denote by $X_{mi}^{(j)}$ the number of generators in $S^{(j)}$ that are contiguous to P_i in V_m, we have

(A1.9) $$X_{mi} = \sum_{j=1}^{8} X_{mi}^{(j)} \qquad (i \le m).$$

We estimate the expected value of $X_{mi}^{(j)}$ as follows. For notational simplicity, we write $P_i=Q_0$, $S^{(j)}=S_0$ and $X_{mi}^{(j)}=X$. Two neighboring sectors of S_0 are denoted by S_1 and S_2 (Fig. A1.3) and, for $j=0,1,2$, $S_j(r)$ designates the region in S_j that is within the distance of r from Q_0. For $j=1,2$, let Q_j denote the generator $P_{i(j)}$ $(1 \le i(j) \le m)$, if any, which is the nearest to Q_0 among the generators contained in S_j and put $r_j=Q_0Q_j$. Note that the angle $Q_1Q_0Q_2$ lies between $\pi/4$ and $3\pi/4$.

From Lemma A1.1 and Lemma A1.2, it follows that if a generator $P_j \in S_0$ $(1 \le j \le m)$ is contiguous to Q_0 in V_m, it must lie in $S_0(R)$, where

(A1.10) $$R^2 = 2(r_1^{\,2} + r_2^{\,2} + \sqrt{2}r_1r_2).$$

Thus the expected value of X conditional on $R=r$ is bounded by $N(r)$, the expected value, conditional on r, of the number of generators of V_m contained in $S_0(r)$:

(A1.11) $$E[X|r] \le N(r).$$

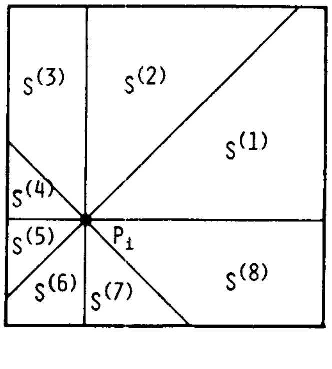


Fig. A1.2. Partition of S into 8 sectors

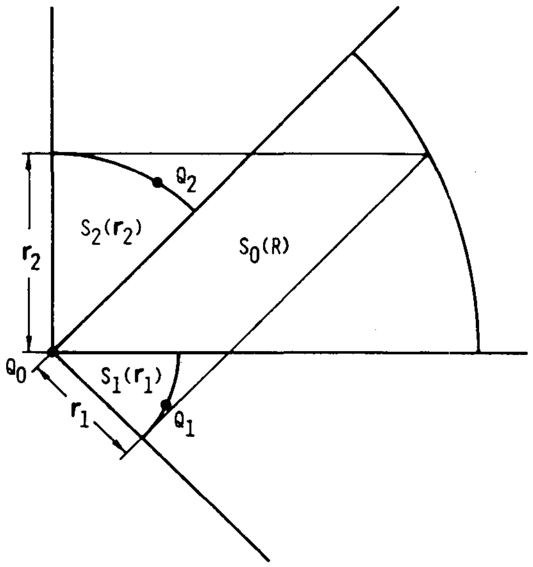


Fig. A1.3. Neighboring sectors

For R defined by (A1.10), we put

(A1.12) $$F(r) = \Pr\{R > r\}.$$

Suppose we have a nonincreasing function $G(r)$ with $G(+\infty)=0$ such that

(A1.13) $$F(r) \le G(r) \quad \text{for } r\ge 0,$$

as well as a nondecreasing function $B(r)$ such that

(A1.14) $$N(r) \le B(r) \quad \text{for } r\ge 0,$$

and

(A1.15) $$\int B(r)\,|dG(r)| \le C_2/8$$

with a constant C_2. Note that (A1.13) implies

(A1.16) $$\int B(r)\,|dF(r)| \le \int B(r)\,|dG(r)|$$

since $B(r)$ is nondecreasing and $B(0) \ge 0$. Then we have

(A1.17) $$\begin{aligned} E[X] &= \int E[X|r]\,|dF(r)| \\ &\le \int N(r)\,|dF(r)| && \text{(by (A1.11))} \\ &\le \int B(r)\,|dF(r)| && \text{(by (A1.14))} \\ &\le \int B(r)\,|dG(r)| && \text{(by (A1.16))} \\ &\le C_2/8. && \text{(by (A1.15))} \end{aligned}$$

Combining (A1.9) and (A1.17), we can obtain (A1.2).

A1.4. Derivation of (A1.15)

In what follows, we will establish (A1.15) under the simplifying assumption that the sectors S_0, S_1 and S_2 may be regarded as unbounded regions. In other words, the generators are assumed as points of a homogeneous planar Poisson process of intensity n and the whole plane is divided into cells of side 2^{-M}.

First we consider the bound $B(r)$ of (A1.14). When $h \le M-1$, each supercell contains at most one generator of V_m, and therefore $N(r)$ is bounded by the number of those supercells of side c which have a nonempty intersection with $S_0(r)$; the latter being bounded by

(A1.18) $$a_1(r/c+b_1)^2,$$

where $a_1 = \pi/8$ and $b_1 = \sqrt{2} + 2(2+\sqrt{2})^{1/2}$. When $h=M$, $N(r)$ is bounded by the expected value, conditional on $R=r$, of the number of generators contained in $S_0(r)$, which is equal to area $(S_0(r))$ multiplied by n, since $S_0(r)\cap(S_1(r_1)\cup S_2(r_2))=\emptyset$ and the generators are subject to the Poisson point process. Thus we have, in view of (A1.3),

(A1.19) $$N(r) \le n\pi r^2/8 \le a_2(r/c)^2,$$

where $a_2 = \pi/(2\alpha^2)$. From (A1.18) and (A1.19), we obtain (A1.14) in either case by choosing

(A1.20) $$B(r) = \max(a_1, a_2)\cdot(r/c+b_1)^2$$

Next, $G(r)$ in (A1.13) is constructed as follows. From (A1.10) it follows that $R>r$ implies $\max(r_1,r_2)>b_2r$, where $b_2 = (4+2\sqrt{2})^{-1/2}$, and therefore

(A1.21) $$F(r) \le \Pr\{r_1>b_2r\} + \Pr\{r_2>b_2r\}.$$

Since $r_j>b_2r$ implies that $S_j(b_2r)$ contains no generator for $j=1,2$ and since $S_j(b_2r)$ includes at least $a_1(b_2r-2b_1c)^2/(2c)^2$ supercells of side $2c$, we have

(A1.22) $$\begin{aligned}\Pr\{r_j>b_2r\} &\le \Pr\{S_j(b_2r) \text{ is empty}\}\\ &= \exp[-na_1(b_2r-2b_1c)^2] \qquad (b_2r>2b_1c;\ j=1,2).\end{aligned}$$

Therefore we have

(A1.23) $$\Pr\{r_j>b_2r\} \le g(r/c) \qquad (r/c > b_3;\ j=1,2),$$

where $g(t) = \exp[-a_3(t-b_3)^2]$, $a_3 = a_1b_2/\alpha^2$ and $b_3 = 2b_1/b_2$, since $nc^2 \ge 1/\alpha^2$ for $h\le M$. Combining (A1.21) and (A1.23) we obtain (A1.13) by setting

(A1.24) $$G(r) = \begin{cases} 2 & \text{if } 0 \le r/c \le b_3,\\ 2\,g(r/c) & \text{if } b_3 < r/c.\end{cases}$$

For $B(r)$ of (A1.20) and $G(r)$ of (A1.24), inequality (A1.15) holds with

(A1.25) $$C_2 = 16\max(a_1,a_2)\int_0^\infty((x/a_3)^{1/2}+b_1+b_3)^2\exp(-x)\,dx.$$

Note that C_2 given in (A1.25) is a constant depending only on α.

Appendix 2. Sketch of the Proof That the Divide-and-Conquer Algorithm Needs $\Omega(n \log n)$ Time Even on the Average

Here we will demonstrate by asymptotic theoretical argument as well as by experiment that the divide-and-conquer algorithm takes $\Omega(n \log n)$ time even on the average.

Let n be the number of generators and set $n_p = n2^{-p}$ $(1 \le p < \log_2 n)$. At the p-th stage of the divide-and-conquer algorithm, 2^{p-1} pairs of right and left diagrams, each containing n_p generators, are to be merged to 2^{p-1} diagrams containing $2n_p$ generators. Denote by L_p the total number of Voronoi edges created at the p-th stage, i.e., the total number of line segments contained in the 2^{p-1} dividing lines (merge curves) created at the p-th stage. The total number L_T of Voronoi edges created by the algorithm in the whole process is then given by

$$L_T = \sum_{p=1}^{\log n} L_p . \tag{A2.1}$$

In the following, we will show that, if the n generators are distributed uniformly in the unit square, the expected value of L_p is bounded from below by $(1/6)n$, i.e.,

$$\mathrm{E}[L_p] \ge (1/6)n \tag{A2.2}$$

for p such that

$$(4/6) \log n < p < (5/6) \log n , \tag{A2.3}$$

when n is large enough. Note that n_p tends to infinity if n tends to infinity with p satisfying (A2.3). Then it follows from (A2.1) and (A2.2) that

$$\mathrm{E}[L_T] \ge (1/36)n \log n , \tag{A2.4}$$

which implies, together with the obvious upper bound $L_T = O(n \log n)$, that

$$\mathrm{E}[L_T] = \Theta(n \log n).$$

Suppose that n generators are distributed randomly uniformly in the unit square and consider a pair of left and right diagrams to be merged, each containing n_p generators. Let $\{P_i \mid i \in L\}$ (resp. $\{P_j \mid j \in R\}$) be the set of generators forming the left diagram V_L (resp. right diagram V_R). The width w_L and w_R of the band regions for the respective diagrams (Fig. A2.1) are random variables with asymptotic mean $n_p/n = 2^{-p}$ and asymptotic variance $(1-n_p/n)n_p/n^2$.

(It is possible to derive the joint distribution of (w_L, w_R) from that of the order statistics, but we omit it.) Note that the n_p generators are distributed uniformly in each of the band regions.

For each P_i $(i \in L)$, let us denote by $P_{N(i)}$ $(N(i) \in L \cup R)$ the point among the $2n_p-1$ generators in $(L \cup R)\backslash\{i\}$ that is nearest to P_i. In case $N(i)$ belongs to R, a portion of the perpendicular bisector of $P_iP_{N(i)}$ constitutes a new Voronoi edge in the merged diagram $V_{L \cup R}$. That is, the number of the Voronoi edges created in merging the two diagrams is not less than the number of left generators P_i such that $N(i) \in R$.

Let f be the probability that $N(i) \in R$ for a fixed i $(\in L)$. Evidently, it may be assumed that this probability does not depend on index i. Then the expected number of the new Voronoi edges created in merging a pair of diagrams of size n_p is asymptotically greater than fn_p, and, since we have 2^{p-1} merging pairs at the p-th stage, we have

(A2.5) $$E[L_T] \geq 2^{p-1}\, f\, n_p = f\, n/2.$$

Next, we will estimate the probability f. Since the band width w_L or w_R is of the order $2^{-p} < n^{-2/3}$, the n_p generators in L can be regarded as being distributed along a line with mean distance $1/n_p$. For p in the range of (A2.3), we have

$$w_L,\ w_R \ll 1/n_p \qquad (p \geq (4/6)\log n > (1/2)\log n\ ),$$

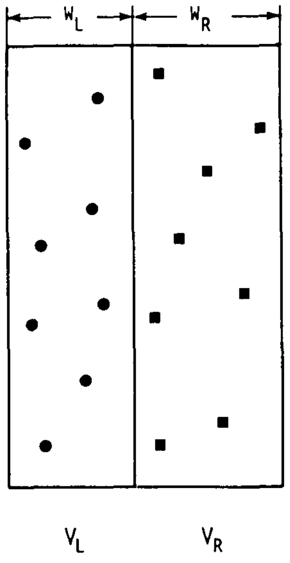


Fig. A2.1. Neighboring band regions to be merged

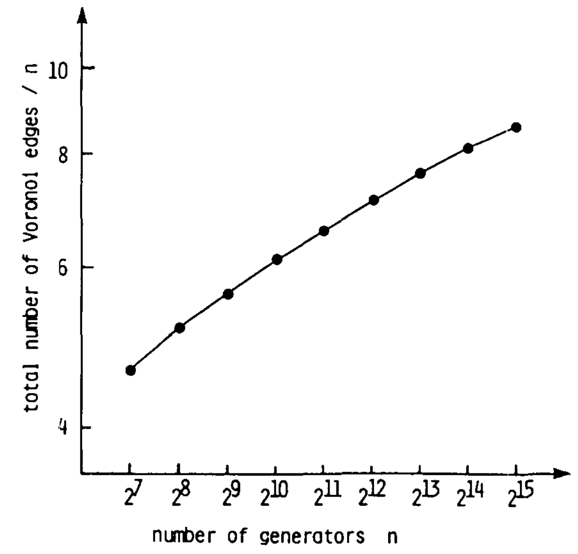


Fig. A2.2. Total number of Voronoi edges created by the divide-and-conquer algorithm

which implies that the band width hardly affects the distance from P_i to other generators in $L \cup R$. In other words, the probability that $N(i)$ belongs to R comes close to 1/2 as n gets large with p satisfying (A2.3). To be specific, we may claim that

(A2.6) $\qquad f \geq 1/3$

for n large. Combining (A2.5) and (A2.6), we obtain (A2.2), and consequently, that the expected total number L_T of the Voronoi edges created by the divide-and-conquer algorithm is of the order $\Theta(n \log n)$.

The total number L_T of Voronoi edges created by the divide-and-conquer algorithm was observed in our experiment. Note that L_T does not depend on a particular implementation but only on the algorithm. The average of L_T/n of 10 independent samples against $n=2^7$ to 2^{15} is plotted in Fig. A2.2, and the minimum and the maximum of L_T among those of the 10 samples, along with the average normalized by n and by $n \log n$, is listed in Table A2.1. As is seen from the table, the behavior of the average of $L_T/(n \cdot \log_2 n)$ can be accounted for by an experimental formula

(A2.7) $\qquad L_T/(n \cdot \log_2 n) = (0.496\, n \cdot \log_2 n + 1.155\, n)/(n \cdot \log_2 n)$

quite well. This fact evidences the theoretical argument that there is a substantial component in L_T, and hence in the computational time of the divide-and-conquer algorithm, which grows as fast as $n \cdot \log n$.

Table A2.1. Total number L_T of Voronoi edges created by the divide-and-conquer algorithm (n=number of generators; 10 samples for each n)

n	Min(L_T)	Max(L_T)	Av.$\frac{(L_T)}{n}$	Av.$\frac{(L_T)}{n \cdot \log_2 n}$	Formula (A2.7)
128	577	608	4.63	0.661	0.661
256	1299	1338	5.14	0.643	0.640
512	2857	2898	5.62	0.624	0.624
1024	6236	6327	6.13	0.613	0.612
2048	13492	13637	6.63	0.603	0.601
4096	28689	29230	7.09	0.592	0.592
8192	61838	62496	7.61	0.586	0.584
16384	132768	133190	8.12	0.580	0.579
32768	280040	282406	8.60	0.573	0.573

References

[1] Bentley, J. L., Weide, B. W., and Yao, A. C.: Optimal Expected-Time Algorithms for Closest Point Problems. *ACM Transactions on Mathematical Software,* Vol.6 (1980), 563-580.

[2] Green, P.J., and Sibson, R.: Computing Dirichlet Tessellation in the Plane. *The Computer Journal,* Vol.21 (1978), 168-173.

[3] Horspool, R. N.: Constructing the Voronoi Diagram in the Plane. *Technical Report SOCS* 79.12, McGill University, 1979.

[4] Imai, H., Iri, M., and Murota, K.: Voronoi Diagram in the Laguerre Geometry and Its Applications. To appear in *SIAM Journal on Computing,* Vol.14, No.1 (1985).

[5] Iri, M., Murota, K., and Matsui, S.: Linear-Time Approximation Algorithms for Finding the Minimum-Weight Perfect Matching on a Plane. *Information Processing Letters,* Vol.12 (1981), 206-209.

[6] Iri, M., Murota, K., and Matsui, S.: Heuristics for Planar Minimum Weight Perfect Matchings. *Networks,* Vol.13 (1983), 67-92.

[7] Iri, M., Murota, K., and Ohya, T.: Geographical Optimization Problems and Their Practical Solutions (in Japanese). *Proceedings of the 1983 Spring Conference of the Operations Research Society of Japan,* 1983, C-2, 92-93.

[8] Iri, M., et al.: Fundamental Algorithms for Geographic Data Processing (in Japanese). *Technical Report* T-83-1, Operations Research Society of Japan, 1983.

[9] Lee, D. T., and Drysdale, R. L., III: Generalization of Voronoi Diagrams in the Plane. *SIAM Journal on Computing,* Vol.10 (1981), 73-87.

[10] Lipton, R. J., and Tarjan, R. E.: Applications of a Planar Separator Theorem. *SIAM Journal on Computing,* Vol.9 (1980), 615-627.

[11] Ohya, T.: Robustness of the Fast Voronoi Algorithm against Nonuniform Distribution of Points (in Japanese). *Proceedings of the 1983 Spring Conference of the Operations Research Society of Japan,* 1983, C-1, 90-91.

[12] Ohya, T., Iri, M., and Murota, K.: An Improved Algorithm for Constructing the Voronoi Diagram (in Japanese). *Proceedings of the 1982 Fall Conference of the Operations Research Society of Japan,* 1982, E-8, 152-153.

[13] Ohya, T., Iri, M., and Murota, K.: A Fast Voronoi-Diagram Algorithm with Quaternary Tree Bucketing. *Information Processing Letters,* Vol.18 (1984) 227-231.

[14] Ohya, T., and Murota, K.: Algorithms and Data Structures for Constructing the Voronoi Diagram (in Japanese). *Proceedings of the 1982 Spring Conference of the Operations Research Society of Japan,* 1982, 2C-3, 185-186.

[15] Shamos, M. I.: *Computational Geometry*. Ph.D. Thesis, Yale University, 1978.

[16] Shamos, M. I., and Hoey, D.: Closest-Point Problems. *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science,* New York, 1975, 151-162.

Takao OHYA: Central Research Institute of Electric Power Industry, Ohtemachi, Chiyoda-ku, Tokyo 100, Japan

Masao IRI: Department of Mathematical Engineering and Instrumentation Physics, Faculty of Engineering, University of Tokyo, Hongo, Bunkyo-ku, Tokyo 113, Japan

Kazuo MUROTA: Institute of Socio-Economic Planning, University of Tsukuba, Sakura, Niihari, Ibaraki 305, Japan