A FAST VORONOI-DIAGRAM ALGORITHM WITH APPLICATIONS TO GEOGRAPHICAL OPTIMIZATION PROBLEMS

Masao Iri
University of Tokyo, Tokyo, Japan
Kazuo Murota
University of Tsukuba, Ibaraki, Japan
Takao Ohya
Central Research Institute of Electric Power Industry, Tokyo, Japan

Abstract There are many kinds of facility location problems, or "geographical optimization problems", which are appropriately formulated in terms of the Voronoi diagram. Except few quite special problems, we can get a solution to a problem of that kind only by means of numerical approach which involves a large number of function evaluations, where each evaluation requires constructing the Voronoi diagram for a tentative distribution of facilities and computing integrals with reference to that diagram. In such a case, the practical feasibility of the numerical solution of the problem depends largely upon the efficiency of the algorithm to be used for constructing the Voronoi diagram. In this paper, we shall formulate a class of location problems and show that, if we use the Voronoi-diagram algorithm recently proposed by the authors, we can numerically solve considerably large problems within a practicable time.

1. Introduction

The location problem has many difficult "faces", among which the "combinatorial face" would be the most essential. However, when we deal with the problem in the continuum, i.e., in the two-dimensional Euclidean plane, we may encounter another, no less difficult, face of it. In fact, the objective function to be minimized or maximized of a problem is often defined in terms of the Voronoi diagram for the given location of facilities (or the like), i.e. the partition of the entire plane into the territories of the facilities. Naturally, we have to resort to some numerical approach to the solution, except for very few particular cases, and the numerical solution will ordinarily consist of iterative evaluation of the objective function as well as of its partial derivatives.

If, as is widely believed, the construction of the Voronoi diagram for given n points in the plane were itself a big computational problem for n large, an algorithm which calls a subroutine for the Voronoi-diagram construction many times would be far from being practical. However, recent progress in computational geometry has produced a number of fast algorithms for the Voronoi diagram, which gives us a bright prospect for the possibility of bringing the numerical solution of the location problem into the practically tractable family.

In the following, we shall formulate a fundamental class of location problems, or "geographical optimization problems", and show that they can be solved numerically within a practicable time for as many as over one hundred points or facilities, if we use a Voronoi-diagram algorithm which is fast enough. The algorithm which the authors recently proposed and which they call the "quaternary incremental method" [14] was found to be qualified as such one.

2. Mathematical and Computational Background for the Voronoi Diagram

For n distinct points $\mathbf{x}_i=[x_i^K]$ $(i=1,\ldots,n)$ given in the N-dimensional Euclidean space $\mathbf{R}^N$ $(K=1,\ldots,N)$, we define a partition of $\mathbf{R}^N$ into n convex polyhedral regions V_i $(i=1,\ldots,n)$ in such a way that every point $\mathbf{x}=[x^K]$ in V_i is closer to $\mathbf{x}_i$ than to any other $\mathbf{x}_j$ $(j\neq i)$:

$$V_i = \bigcap_{j:j\neq i} \{\mathbf{x} \mid \|\mathbf{x}-\mathbf{x}_i\| < \|\mathbf{x}-\mathbf{x}_j\|\}. \tag{2.1}$$

Obviously we have

$$\overline{\bigcup_i V_i} = \mathbf{R}^N. \tag{2.2}$$

V_i is a kind of "territory" of point $\mathbf{x}_i$ and is called the Voronoi region belonging to $\mathbf{x}_i$. The partition determines in an obvious manner a polyhedral complex, which is called the Voronoi diagram for the given n points $\mathbf{x}_i$'s. We shall denote the common boundary of V_i and V_j by

$$W_{ij} = \partial V_i \cap \partial V_j. \tag{2.3}$$

In the two-dimensional case, i.e. in $\mathbf{R}^2$, the one-dimensional section of a Voronoi diagram is a planar graph, of which the vertices we shall call the Voronoi points and the edges the Voronoi edges. The points $\mathbf{x}_i$'s will be called generators. A Voronoi edge is part of the perpendicular bisector of some couple of generators, and a Voronoi point is the excenter of some triple of generators. Since the degree of a Voronoi point, as a vertex of the graph, is three except for the degenerate case, we may consider the dual graph whose vertices are generators and which

has an edge between two generators iff the corresponding Voronoi regions have a boundary edge in common. The dual graph is called the Delaunay triangulation of $\mathbb{R}^2$.

The Voronoi diagram, as well as the Delaunay triangulation, has now been recognized as a concept of fundamental importance in many kinds of problems in geography, urban planning, environmental control, physics, biology, ecology, etc. [2], [3], [9], [16]. Many computational algorithms for constructing the Voronoi diagram have also been proposed, but it has been known that the Voronoi diagrams in a space whose dimensionality is higher than two are rather hard to construct practically, as we are usually concerned with the diagrams with very many points, say more than hundreds or even thousands in number. (See, e.g., [1].) However, if we confine ourselves to the case of two dimensions, a rather primitive incremental algorithm such as the one proposed in [5] sometimes works fairly well, and several "theoretically optimal" algorithms of the divide-and-conquer type with the worst-case time complexity $O(n \log n)$ (and with the linear space complexity $O(n)$) have been proposed, improved and generalized [7], [10], [11], since the appearance of the epoch-making paper [15] in computational geometry.

It is interesting in this context to see that, for almost all problems that we might encounter in practical situations, the more primitive algorithm of the incremental type can be improved to run substantially faster than the theoretically optimal algorithms of the divide-and-conquer type. In fact, the algorithm of the incremental type which makes use of a quaternary tree (which was proposed by the authors [13], [14] and will be outlined in Appendix) has been proved to run in linear time $O(n)$ on the average——more specifically, in about 2.5 ms per point on a typical large digital computer——and to be quite robust against nonuniformity of the distribution of points [14].

3. Problem Formulation

To be specific, let us consider n "facilities" placed, respectively, at points $\mathbf{x}_1,\ldots,\mathbf{x}_n$ of the N-dimensional Euclidean space $\mathbb{R}^N$ ($\mathbf{x}_i=[x_i^\kappa]$, $\kappa=1,\ldots,N$). The territories of those facilities are the Voronoi regions V_i defined by (2.1). We consider furthermore a distribution of "inhabitants" represented by a certain measure $d^N\mu(\mathbf{x})$ on $\mathbb{R}^N$, to be called the measure of population density.

It will not be very unrealistic to assume that an individual inhabitant living in region V_i will enjoy a service of the i-th facility and that the cost for him to gain access to a facility is a function

(ordinarily, monotone increasing, and even convex) of the Euclidean distance of $\mathbf{x}$ and the location $\mathbf{x}_i$ of the nearest facility. Then the total cost connected with the serviceability of the facilities is written qua function of $\mathbf{x}_1,\ldots,\mathbf{x}_n$ as

$$F(\mathbf{x}_1,\ldots,\mathbf{x}_n) = \frac{1}{2}\int_{\mathbb{R}^N} f(\min_j \|\mathbf{x}-\mathbf{x}_j\|^2)\,d^N\mu(\mathbf{x})$$

$$= \frac{1}{2}\sum_{i=1}^{n}\int_{V_i} f(\|\mathbf{x}-\mathbf{x}_i\|^2)\,d^N\mu(\mathbf{x}), \tag{3.1}$$

where the factor 1/2 has been introduced for convenience' sake without loss of generality. Thus we have been led to an optimization problem by means of which we may determine the optimum locations of a given number of facilities for a given population distribution, i.e., the problem of minimizing the function F in (3.1). Since each $\mathbf{x}_i$ has N coordinates x_i^κ ($\kappa=1,\ldots,N$), the problem is essentially the minimization with respect to as many as Nn variables.

4. Solution Algorithm

Before entering into the discussion of a solution algorithm, we should make some observation on the properties of the objective function F of (3.1). Firstly, even if f is convex, F is in general nonconvex, as is evident from the obvious fact that F has several local minima and/or stationary points. Therefore, unless we dare to tackle the combinatorial difficulty connected with the existence of many local minima of a nonconvex function, we have to be contented with "a local minimum". Secondly, F is in general nondifferentiable, although the value of F will depend smoothly on $\mathbf{x}_i$'s for smooth f so long as the locations $\mathbf{x}_i$ of the generators (facilities) are varying in such a way that the topology of the Voronoi diagram may remain unchanged. (This will be seen more concretely in terms of the explicit formula for the partial derivatives of F which we shall calculate in the following.) We can expect at best the existence of its subgradient. Since no general optimization algorithm is available at present which works for such functions better than the most primitive class of descent methods, we have to resort to a variant of primitive descent algorithm.

Thus, we shall investigate the algorithms of the following type, where we denote by $\mathbf{X}$ the Nn-dimensional unknown vector whose components are the components of n vectors $\mathbf{x}_1,\ldots,\mathbf{x}_n$:

$$\mathbf{X}^T = [\mathbf{x}_1^T, \mathbf{x}_2^T, \ldots, \mathbf{x}_n^T]\,. \tag{4.1}$$

<0> Initial guess:——Start from an arbitrarily given initial guess $\mathbf{X}^{(0)}$, and repeat <1>-<3> for $\nu=0,1,2,\ldots$ until some stopping criterion is satisfied.

<1> Search direction:——Compute the gradient $\nabla_{\mathbf{X}}F(\mathbf{X}^{(\nu)})$, or the partial derivatives, of F at the ν-th approximate solution $\mathbf{X}^{(\nu)}$. Then determine the search direction $\mathbf{d}^{(\nu)}$ from $\nabla_{\mathbf{X}}F(\mathbf{X}^{(\nu)})$ (and from some auxiliary quantities if we want).

<2> Line search:——Determine $\hat{\alpha}$ (up to a certain degree of approximation) such that

$$F(\mathbf{X}^{(\nu)} + \hat{\alpha}\,\mathbf{d}^{(\nu)}) = \min_{\alpha} F(\mathbf{X}^{(\nu)} + \alpha\,\mathbf{d}^{(\nu)}). \tag{4.2}$$

<3> New approximation:——Set

$$\mathbf{X}^{(\nu+1)} = \mathbf{X}^{(\nu)} + \hat{\alpha}\,\mathbf{d}^{(\nu)}. \tag{4.3}$$

There are a number of variants of the algorithm of the above type corresponding to the choice of the search direction in <1>, the manner of performing the line search in <2> and the rule for stopping the iteration. We have tested several variants which will be described in section 6.

5. Calculation of the Partial Derivatives

Since the objective function in (3.1) is not familiar in form it may not be useless to write down its partial derivatives (the gradient and the Hessian). In so doing, we adopt the tensor notation in $\mathbf{R}^N$ in order to maintain the geometrical meanings of the relevant expressions as clear as possible. Thus, we denote by $g_{\lambda\kappa}$ the metric tensor in $\mathbf{R}^N$ (which may be regarded simply as the N×N identity matrix if one would not like to be involved in tensor notation) and use Einstein's summation convention (i.e., if one and the same index appears at two places of a term, one as a contravariant (upper) index and the other as a covariant (lower) index, then the summation with respect to that index over the range $\{1,2,\ldots,N\}$ is understood). For example, we write, for two vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbf{R}^N$,

$$\|\mathbf{x} - \mathbf{y}\|^2 = g_{\lambda\kappa}x^{\kappa}y^{\lambda} \quad (= \sum_{\lambda=1}^{N}\sum_{\kappa=1}^{N} g_{\lambda\kappa}x^{\kappa}y^{\lambda}) \ . \tag{5.1}$$

We need some more auxiliary symbols to make the expressions handsome. The distance between two generators $\mathbf{x}_i$ and $\mathbf{x}_j$ will be denoted by

$$\alpha_{ij} \equiv \|\mathbf{x}_i - \mathbf{x}_j\| \ , \tag{5.2}$$

and the (N-1)-dimensional measure induced by $d^N\mu(\mathbf{x})$ on a hypersurface in $\mathbf{R}^N$ will be denoted by $d^{N-1}\mu(\mathbf{x})$.

The partial derivative of the F of (3.1) with respect to an x_i^{λ} will

consist of two components, one coming from the derivative of the integrand function and the other from the variation of the region V_i itself as well as of the regions V_j's adjacent to V_i due to the variation of $\mathbf{x}_i$. The first component takes the form:

$$\frac{1}{2}\int_{V_i}\frac{\partial}{\partial x_i^\lambda} f(\|\mathbf{x}-\mathbf{x}_i\|^2)\,d^N\mu(\mathbf{x}) = \int_{V_i} g_{\lambda\kappa}(x_i^\kappa-x^\kappa) f'(\|\mathbf{x}-\mathbf{x}_i\|^2)\,d^N\mu(\mathbf{x}) \tag{5.3}$$

and the second component is the sum, taken over all the boundary hyperplanes W_{ij}'s of V_i, of the terms of the form due to the variation of the V_i:

$$\int_{W_{ij}} f(\|\mathbf{x}-\mathbf{x}_i\|^2)\frac{1}{\alpha_{ij}} g_{\lambda\kappa}(x_i^\kappa-x^\kappa)\,d^{N-1}\mu(\mathbf{x}) \tag{5.4}$$

and the terms due to the variation of an adjacent region V_j:

$$-\int_{W_{ij}} f(\|\mathbf{x}-\mathbf{x}_j\|^2)\frac{1}{\alpha_{ij}} g_{\lambda\kappa}(x_i^\kappa-x^\kappa)\,d^{N-1}\mu(\mathbf{x}). \tag{5.5}$$

However, since $\|\mathbf{x}-\mathbf{x}_j\|=\|\mathbf{x}-\mathbf{x}_i\|$ on W_{ij}, the terms constituting the second component cancel themselves, so that only the first component will remain, i.e., we have

$$\frac{\partial F}{\partial x_i^\lambda} = \int_{V_i} g_{\lambda\kappa}(x_i^\kappa-x^\kappa) f'(\|\mathbf{x}_i-\mathbf{x}\|^2)\,d^N\mu(\mathbf{x}). \tag{5.6}$$

The second derivatives of F may be calculated in a similar but rather complicated way to give

$$\frac{\partial^2 F}{\partial x_i^\kappa \partial x_j^\lambda} = \begin{cases} H^i_{\lambda\kappa} + G^i_{\lambda\kappa} & \text{for } j=i, \\ G^{ji}_{\lambda\kappa} & \text{for } j\neq i \text{ with } W_{ij}\neq\emptyset, \\ 0 & \text{for } j\neq i \text{ with } W_{ij}=\emptyset, \end{cases} \tag{5.7}$$

where

$$H^i_{\lambda\kappa}=\int_{V_i}[g_{\lambda\kappa}f'(\|\mathbf{x}-\mathbf{x}_i\|^2)+2g_{\lambda\nu}g_{\kappa\mu}(x_i^\mu-x^\mu)(x_i^\nu-x^\nu)f''(\|\mathbf{x}-\mathbf{x}_i\|^2)]\,d^N\mu(\mathbf{x}),$$

$$G^{ji}_{\lambda\kappa} = K^{jji}_{\lambda\kappa}, \qquad G^i_{\lambda\kappa} = -\sum_{\substack{j:j\neq i \\ W_{ij}\neq\emptyset}} K^{iji}_{\lambda\kappa},$$

$$K^{kji}_{\lambda\kappa}=\int_{W_{ij}}\frac{1}{\alpha_{ij}} g_{\lambda\nu}g_{\kappa\mu}(x_i^\mu-x^\mu)(x_k^\nu-x^\nu)f'(\|\mathbf{x}-\mathbf{x}_i\|^2)\,d^{N-1}\mu(\mathbf{x}). \tag{5.8}$$

The special case where $f(t)=t$ is worth noting, because, in that case, the partial derivatives of F are written in a form which admits a

direct physical interpretation as follows. For the components of the gradient of F, we have

$$\frac{\partial F}{\partial x_i^\lambda} = \int_{V_i} g_{\lambda\kappa}(x_i^\kappa - x^\kappa)\, d^N\mu(\mathbf{x}) = \mu(V_i)\cdot g_{\lambda\kappa}(x_i^\kappa - \bar{x}_i^\kappa), \tag{5.9}$$

where

$$\mu(V_i) = \int_{V_i} d^N\mu(\mathbf{x}) \tag{5.10}$$

is the total measure of the region V_i, and

$$\bar{x}_i^\kappa = \int_{V_i} x^\kappa d^N\mu(\mathbf{x})/\mu(V_i) \tag{5.11}$$

is the "centroid" of V_i with respect to the measure $d^N\mu(\mathbf{x})$. For the components of the Hessian, we have

$$H^i_{\lambda\kappa} = \mu(V_i)\cdot g_{\lambda\kappa}, \tag{5.12.1}$$

$$K^{kji}_{\lambda\kappa} = \frac{\mu_{ij}(W_{ij})}{\alpha_{ij}}\, g_{\lambda\nu}g_{\kappa\mu}[x_i^\mu x_k^\nu - x_i^\mu \bar{x}_{ij}^\nu - \bar{x}_{ij}^\mu x_k^\nu + \overline{xx}_{ij}^{\mu\nu}], \tag{5.12.2}$$

where

$$\mu_{ij}(W_{ij}) = \int_{W_{ij}} d^{N-1}\mu(\mathbf{x}) \tag{5.13}$$

is the total measure of W_{ij} with respect to $d^{N-1}\mu(\mathbf{x})$,

$$\bar{x}_{ij}^\mu = \int_{W_{ij}} x^\mu d^{N-1}\mu(\mathbf{x})/\mu_{ij}(W_{ij}) \tag{5.14}$$

is the centroid of W_{ij} with respect to the measure $d^{N-1}\mu(\mathbf{x})$, and

$$\overline{xx}_{ij}^{\mu\nu} = \int_{W_{ij}} x^\mu x^\nu d^{N-1}\mu(\mathbf{x})/\mu_{ij}(W_{ij}) \tag{5.15}$$

is the mean second moment of W_{ij} with respect to $d^{N-1}\mu(\mathbf{x})$.

6. Numerical Examples

Choosing the simplest case of $f(t)=t$, we performed a number of experimental computations for two-dimensional problems (N=2) having generators $n=16$ to 256 in number, with respect to three different population distributions, by means of various algorithms with two different search directions and two different line search methods. We considered the following population distributions confined in the unit square $\mathbf{S}=[-\frac{1}{2}, \frac{1}{2}]\times[-\frac{1}{2}, \frac{1}{2}]$:

1° $d^2\mu(\mathbf{x})$ is uniform in $\mathbb{S}$ and vanishes outside;

2° $d^2\mu(\mathbf{x})$ is proportional to $\exp(-25\|\mathbf{x}\|^2)$ in $\mathbb{S}$ and vanishes outside (Tanner-Sherratt type distribution of the urban population [12]);

3° $d^2\mu(\mathbf{x})$ is proportional to $\exp(-\|\mathbf{x}\|\cdot(25\|\mathbf{x}\|-10))$ in $\mathbb{S}$ and vanishes outside (Newling type population distribution [12]).

(See Fig. 1 for these distributions.)

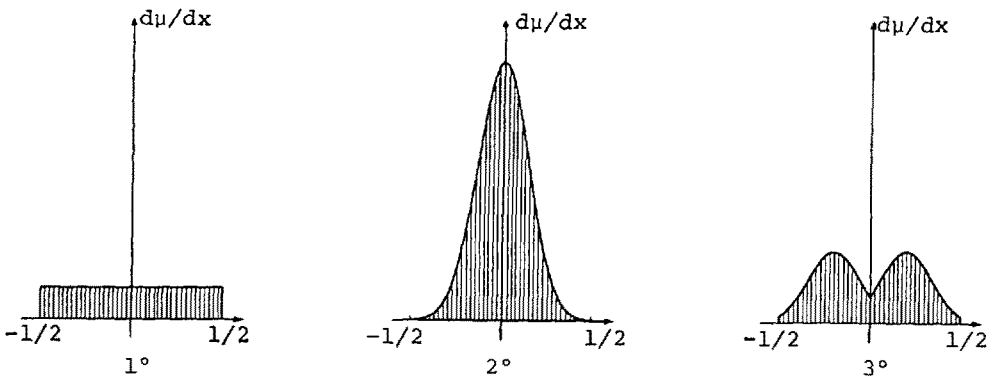


Fig.1. Population distributions used in the experiments (one-dimensional section)

In Fig. 2, examples for n=128 are shown. Starting from a random initial distribution of n generators such as shown in Fig. 2(a) together with the corresponding Voronoi diagram, we had, after a sufficiently large number of iterations, the configurations shown in Figs. 2(b)-(d), respectively, for the population distributions of types 1°-3°. (The number of iterations depends on the variant of the algorithm used; see Fig. 3.)

As we mentioned in Introduction, we used a fast Voronoi diagram algorithm we recently proposed [14] for the construction of the Voronoi diagram. As the criterion for terminating the iteration the condition:

$$\max_{i,\kappa} |x_i^{(\nu+1)\kappa} - x_i^{(\nu)\kappa}| \leq 10^{-5} \tag{6.1}$$

was adopted in the experimental computation.

For the line search, we tested two variants of the so-called "Goldstein method" [4], which chooses $\hat{\alpha}$ so as to satisfy the inequalities

$$\mu_2 \hat{\alpha} \mathbf{d}^{(\nu)} \cdot \nabla_{\mathbf{x}} F(\mathbf{x}^{(\nu)}) \leq F(\mathbf{x}^{(\nu)} + \hat{\alpha}\mathbf{d}^{(\nu)}) - F(\mathbf{x}^{(\nu)}) \leq \mu_1 \hat{\alpha} \mathbf{d}^{(\nu)} \cdot \nabla_{\mathbf{x}} F(\mathbf{x}^{(\nu)}) \tag{6.2}$$

with appropriately prescribed parameters μ_1 and μ_2 $(0<\mu_1<\mu_2<1)$. (We adopted the values $\mu_1=0.2$ and $\mu_2=0.9$ throughout our experiment.) In searching for such an $\hat{\alpha}$, we begin with an initial guess $\alpha^{(0)}$, increase

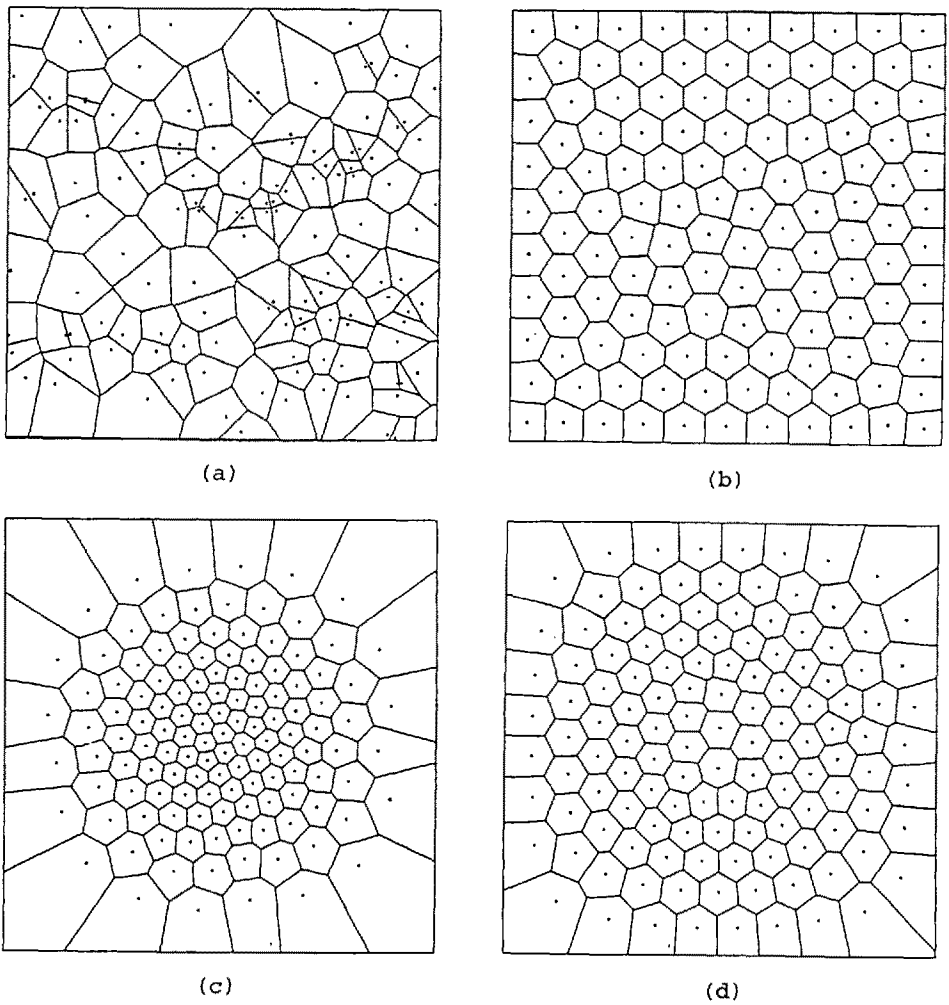


Fig.2. (a) Initial distribution
(b) Near optimum distribution for the population density of type 1°
(c) Near optimum distribution for the population density of type 2°
(d) Near optimum distribution for the population density of type 3°

the value step by step as $\alpha^{(1)}$, $\alpha^{(2)}$,... until the left inequality in (6.2) holds, say at $\alpha^{(m)}$. If $\alpha^{(m)}$ satisfies the right inequality as well, then we stop, setting $\hat{\alpha}=\alpha^{(m)}$. If $\alpha^{(m)}$ does not satisfy the right inequality, then we search for an $\hat{\alpha}$ by means of the binary search in the interval $(\alpha^{(m-1)},\alpha^{(m)})$. Actually we tested the following two kinds of sequences $\alpha^{(0)}$, $\alpha^{(1)}$,...; one is in an "arithmetic progression"

$$\text{A:}\quad \alpha^{(i)} = i\,\alpha^{(0)}, \tag{6.3}$$

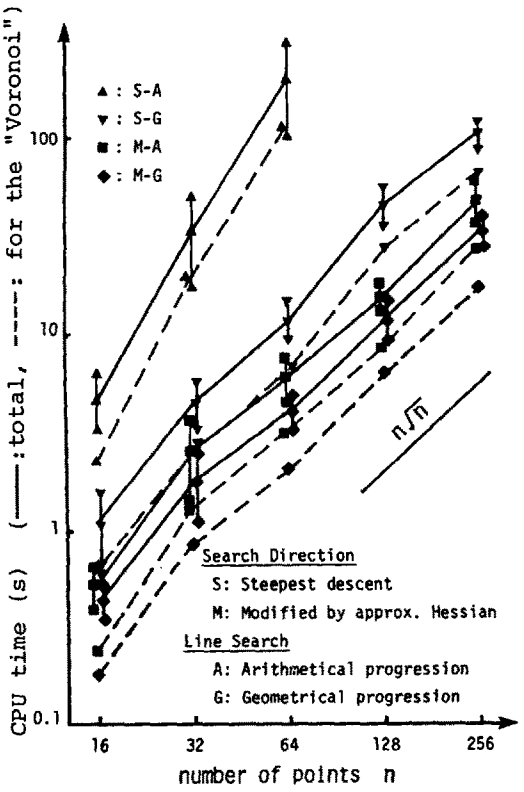


Fig.3. Comparison of the computation times by different solution algorithms

and the other in an "geometric progression"

$$G: \quad \alpha^{(i)} = 2^i \alpha^{(0)}. \tag{6.4}$$

(Incidentally, we set $\alpha^{(0)}=1$ rather arbitrarily, and assumed $\alpha^{(-1)}=0$.) In the following, we shall abbreviate the line search based on (6.3) as "A", and that based on (6.4) as "G".

For the simplest search direction $\mathbf{d}^{(\nu)}$, we took the direction of "steepest descent" (to be abbreviated as "S" in the following"), or more specifically (see (5.9)),

$$S: \mathbf{d}^{(\nu)} = -\nabla_{\mathbf{X}} F(\mathbf{X}^{(\nu)}),$$

$$\mathbf{x}_i^{(\nu+1)} = \mathbf{x}_i^{(\nu)} + \hat{\alpha}(\bar{\mathbf{x}}_i^{(\nu)} - \mathbf{x}_i^{(\nu)}) \cdot \mu(V_i). \tag{6.5}$$

We also investigated a more sophisticated direction $\mathbf{d} = -H^{-1} \cdot \nabla_{\mathbf{X}} F$, which is obtained by modifying the steepest-descent direction with a certain approximation H of the Hessian (2n×2n matrix) of F. Since the exact Hessian (see (5.7) and (5.12)) is too complicated to incorporate in the

iteration process, we adopted the part $H^{i}_{\lambda\kappa}$ (see (5.12.1)) for the "approximation". (It is numerically not a good approximation, but it is qualitatively qualified as such because it is symmetric and positive definite and has an adequate physical dimension.) Specifically, we have from (5.7) and (5.12.1)

$$\text{M:}\quad \mathbf{d}^{(\nu)} = -H^{-1}\cdot\nabla_{\mathbf{x}}F(\mathbf{x}^{(\nu)}),$$

$$\mathbf{x}_i^{(\nu+1)} = \mathbf{x}_i^{(\nu)} + \hat{\alpha}(\bar{\mathbf{x}}_i^{(\nu)} - \mathbf{x}_i^{(\nu)}). \tag{6.6}$$

We shall abbreviate this choice of search direction as "M" (standing for "modified"). It may be interesting to see that the only difference between the steepest-descent direction and the modified direction in this case consists in whether or not there is the factor $\mu(V_i)$ in the modification term of (6.5) and (6.6).

The results of computational experiments for the population density of type 1° are summarized in Fig. 3. (The results for the other population densities looked quite similar.) From these results, we can get the following observations.

(a) From the viewpoint of computational time, the algorithm (M-G) using the approximate Hessian with the geometric-progression line search performs best. (The difference in performance of various algorithms is rather conspicuous.)

(b) Except for the algorithm (S-A) using the steepest-descent direction with the arithmetic-progression line search, the computation time seems to increase with the number of generators in the order of about $n^{3/2}$.

(c) The larger part of computation time is spent for the construction of the Voronoi diagrams.

Furthermore, we counted the number of function evaluations in each line search, and observed that

(d) The average number of function evaluations needed in one line search is O(n) by the algorithm S-A, O(log n) by the algorithm S-G, and a constant independent of n by the algorithms M-A and M-G. (The "constant" in the last case is about 2, and about seven line searches out of ten required indeed less than 3 function evaluations.)

Thus, we may conclude that we can solve the geographical optimization problems (of the kind discussed in the present paper) with as many as hundreds of facilities within a practicable time if we make use of a sufficiently fast Voronoi-diagram algorithm.

Acknowledgement

The authors are indebted to many colleagues and friends for their substantial collaboration in computational experiments and valuable advices from various standpoints. Among them, the authors would like to mention, with cordial gratitude, the names of Prof. D. Avis of McGill University, Prof. T. Koshizuka of the University of Tsukuba, and Prof. A. Okabe, Dr. T. Asano and Mr. H. Imai of the University of Tokyo.

The research was supported in part by the Grant in Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan.

References

[1] Avis, D., and Bhattacharya, B.K.: Algorithms for computing d-dimensional Voronoi diagrams and their duals. Technical Report SOCS 82.5, McGill University, Montreal, Canada, 1982.

[2] Boots, B.N.: Some observations on the structure of socio-economic cellular networks. Canadian Geographer, Vol.19, 1975, pp.107-120.

[3] Finney, J.L.: Random packings and the structure of simple liquids — I. The geometry of random close packing. Proceedings of the Royal Society of London, Vol.A-319, 1970, pp.479-493.

[4] Goldstein, A.A.: Constructive Real Analysis. Harper & Row, 1968.

[5] Green, P.J., and Sibson, R.: Computing Dirichlet tessellation in the plane. The Computer Journal, Vol.21, 1978, pp.168-173.

[6] Horspool, R.N.: Constructing the Voronoi diagram in the plane. Technical Report SOCS 79.12, McGill University, Montreal, Canada, 1979.

[7] Imai, H., Iri, M., and Murota, K.: Voronoi diagram in the Laguerre geometry and its applications. SIAM Journal on Computing (to appear).

[8] Iri, M., Murota, K., and Ohya, T.: Geographical optimization problems and their practical solutions (in Japanese). Proceedings of the 1983 Spring Conference of the OR Society of Japan, C-2, pp.92-93.

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

[10] Lee, D.T.: Two-dimensional Voronoi diagrams in the L_p-metric. Journal of the ACM, Vol.27, 1980, pp.604-618.

[11] Lee, D.T., and Drysdale, R.L.,III.: Generalization of Voronoi diagrams in the plane. SIAM Journal on Computing, Vol.10, 1981, pp.73-87.

[12] Newling, B.E.: The spatial variation of urban population densities. Geographical Reviews, 1969, 4.

[13] Ohya, T., Iri, M., and Murota, K.: A fast incremental Voronoi-diagram algorithm with quaternary tree. (Submitted to Information Processing Letters)

[14] Ohya, T., Iri, M., and Murota, K.: Improvements of the incremental method for the Voronoi diagram with computational comparison of various algorithms. (Submitted to the Journal of the OR Society of Japan)

[15] Shamos, M.I., and Hoey, D.: Closest-point problems. Proceedings of the 16th Annual IEEE Symposium on the Foundations of Computer Scinece, Berkeley, 1975, pp.151-162.

[16] M. Tanemura and M. Hasegawa: Geometrical models of territory I —— Models for synchronous and asynchronous settlement of territories. Journal of Theoretical Biology, Vol.82, 1980, pp.477-496.

Appendix: Quaternary Incremental Algorithm for the Voronoi Diagram

We shall give a brief description of the method of constructing the Voronoi diagram for n points (or, generators) given in a unit square of the Euclidean plane, according to [14]. (See also [13].)

The method is basically of the incremental type [5]; i.e., it starts from the Voronoi diagram $\mathbf{V}_3$ and augments it to $\mathbf{V}_4, \mathbf{V}_5, \ldots$ until the complete diagram $\mathbf{V}_n$ is obtained. In augmenting the diagram $\mathbf{V}_{m-1}$ to $\mathbf{V}_m$ by adding a new generator $\mathbf{x}_m$, we first determine the Voronoi region $V_{N[m]}$ to which $\mathbf{x}_m$ belongs in $\mathbf{V}_{m-1}$ (Phase 1:" Nearest Neighbor Search"). Then we construct the new region V_m in $\mathbf{V}_m$ which is the territory of the new $\mathbf{x}_m$ as follows (Phase 2:"Modification of the Diagram"). (See Fig. A1, and also [5].)

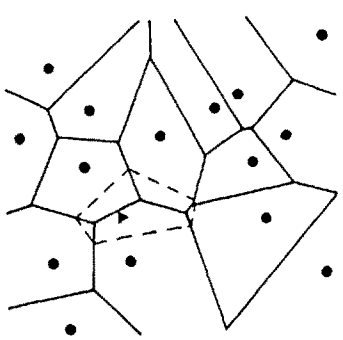


Fig.A1. Construction of the new region for a newly added point

Solid lines: Old diagram
●,●,... : Old generators
▲ : New generator
Broken lines: The boundary of the region for the new generator

Let ℓ_1 be the perpendicular bisector of $\mathbf{x}_m$ and $\mathbf{x}_{N[m]}$. The ℓ_1 intersects two of the boundary edges of $V_{N[m]}$. Let the region adjacent to $V_{N[m]}$ at one of these two edges be $V_{N_1[m]}$ (the territory of generator $\mathbf{x}_{N_1[m]}$), and the perpendicular bisector of $\mathbf{x}_m$ and $\mathbf{x}_{N_1[m]}$ be ℓ_2. The ℓ_2 intersects one of the boundary edges of $V_{N_1[m]}$, other than that which ℓ_1 intersects. Let the region adjacent to $V_{N_1[m]}$ at that edge be $V_{N_2[m]}$ (the territory of $\mathbf{x}_{N_2[m]}$), and the perpendicular bisector of $\mathbf{x}_m$ and $\mathbf{x}_{N_2[m]}$ be ℓ_3. Repeating in this manner, we shall return to the initial region $V_{N[m]}$ to complete the new region of $\mathbf{x}_m$.

What is crucial for the efficiency of the method is (i) in what order to add generators, and (ii) how to find the region $V_{N[m]}$ in the old diagram $\mathbf{V}_{m-1}$ to which the newly added generator $\mathbf{x}_m$ belongs. It may be intuitively not very difficult to admit that, if we augment the diagram keeping it as uniform and similar as possible at every stage, then we have an algorithm which runs in linear time $O(n)$ on the average. In order to realize this uniformity and similarity, we make use of a quaternary structure in the following way.

<1> Divide each side of the unit square into 2^M equal parts to get 4^M small square "buckets", where M is chosen such that $4^M \doteqdot n$.

```
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;
```

Fig.A2. Bucket-numbering algorithm

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

Fig.A3. Example of numbering buckets (M=4)

<2> Number the 4^M buckets (from 0 to 4^M-1) by the "bucket-numbering algorithm" shown in Fig. A2 (see also an example for M=4 in Fig. A3), where it should be noted that this algorithm runs in O(n) time.

<3> Make a quaternary tree, with leaves (ordered from left to right) corresponding to the 4^M buckets (ordered according to the bucket numbers). Associate each generator to the bucket (=leaf) to which it belongs. (This can be done by multiplying the coordinates by 2^M and then truncating the fractional parts off.)

Scanning the leaves from left to right, if a leaf contains a generator, put it in all the ancestors of the quaternary tree in which no generator has yet been put. (See Fig. A4 for <3>.)

<4> Scanning the nodes of the tree starting from the top (root) in the breadth-first manner (from right to left on the same level), as soon as we find a generator i that has not been added to the diagram, add it to the diagram and construct its territory (the new Voronoi region) as has been described in the above (Phase 2).

In Phase 1, adopt the Voronoi region V_j of the generator x_j in the

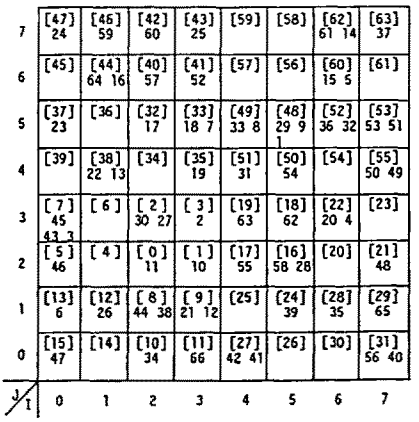
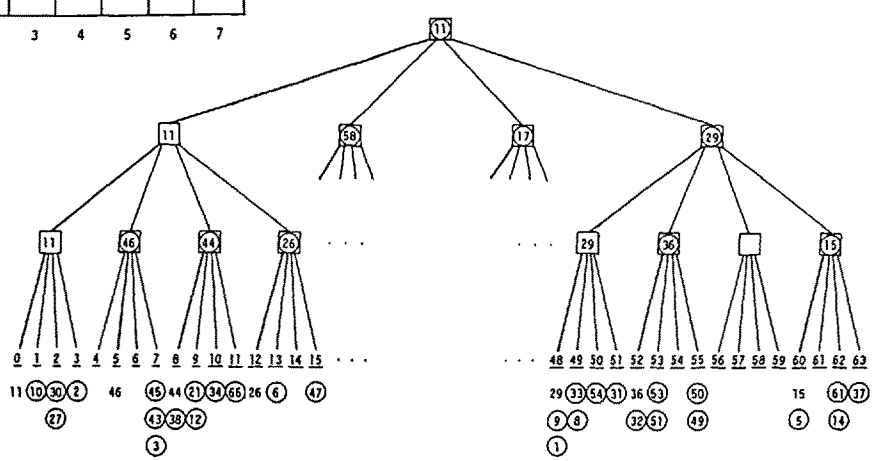


Fig.A4. Example of bucketing 66 generators distributed in the unit square in constructing the corresponding quaternary tree. (Bucket-numbers are indicated in brackets, whereas generator-numbers are indicated without brackets.)



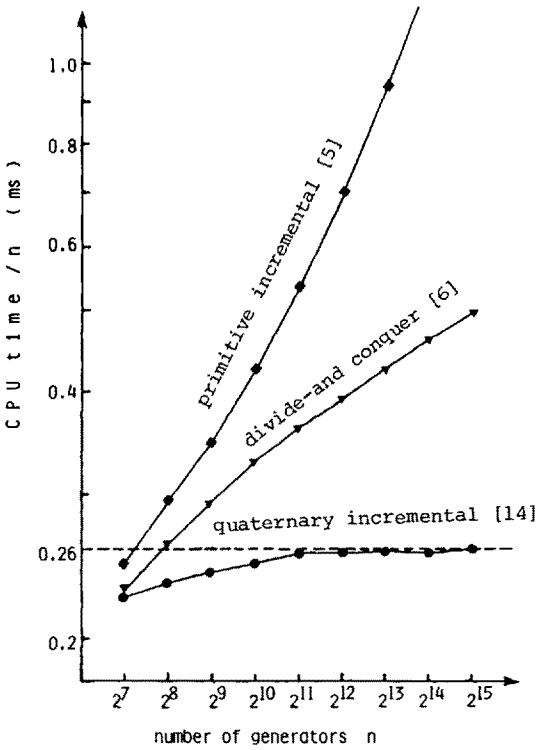


Fig.A5. Comparison of the performances of three different Voronci algorithms

father node as the initial guess for the $V_{N[m]}$ to which the new generator $\mathbf{x}_m$ belongs. (The $V_{N[m]}$ is determined by starting from the V_j and moving from a region to another in such a way that the distance of generator $\mathbf{x}_m$ to the generator of the region may decrease monotone.)