FEM-like Fast Marching Method for the Computation of the Boat-Sail Distance and the Associated Voronoi Diagram

Tetsushi Nishida and Kokichi Sugihara
Department of Mathematical Informatics,
University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
nishida@mist.i.u-tokyo.ac.jp, sugihara@mist.i.u-tokyo.ac.jp

Abstract

A new concept called a boat-sail distance is introduced on the surface of water with flow, and it is used to define a generalized Voronoi diagram, in such a way that the water surface is partitioned into regions belonging to the nearest harbors with respect to this distance. The problem of computing this Voronoi diagram is reduced to a boundary value problem of a partial differential equation, and a numerical method for solving this problem is constructed. The method is a modification of a so-called fast marching method originally proposed for the eikonal equation. Computational experiments show the efficiency and the stableness of the proposal method.

keywords: Boat-sail Distance, Voronoi Diagram, Eikonal Equation, Fast Marching Method, FEM-like scheme

1 Introduction

The concept of the Voronoi diagram is one of the most fundamental data structures in computational geometry, and its algorithms and applications have been studied extensively [6, 9, 15].

The Voronoi diagram has also been generalized in a variety of directions. A typical such direction is the generalization of the distance. The most fundamental Voronoi diagram is defined according to the Euclidean distance, while many generalized Voronoi diagrams are generated by replacing the Euclidean distance with other distances.

The first class of generalized distances are the distances that satisfy the distance axiom. This class includes L_1-distance, L_∞-distance, L_p-distance for $1 < p < \infty$ [13], and convex distances [8, 17]. The second class contains weighted distances such as an additively weighted distance [4], a multiplicatively weighted distance [7], and the Laguerre distance [5, 10]. There are still many other distances. They include a geodesic distance [2, 11, 14], a collision-avoidance distance [3], a distance in a river [20], a skew distance [1], a peeper's distance [6], a crystal-growth distance [12, 16], Karlsruhe distance [11], and the ski distance [11], and the Hausdorff distances [6].

In this paper, we introduce still another generalization of the Voronoi diagram, which we encounter when a boat sails on the surface of water with flow. Suppose that we want to travel on the surface of water with a boat. If there is no flow of water, the boat can move in any direction at the same maximum speed. If the water flows, on the other hand, the speed of the boat is anisotropic; the boat can move faster in the same direction as the flow, while it move only slowly in the direction opposite to the flow direction. Modeling this situation, we introduce a boat-sail distance and define the Voronoi diagram associated with this distances.

The boat-sail Voronoi diagram is a generalization of the Voronoi diagram in a river by Sugihara [20], where the water flows homogeneously, while in this paper we consider an arbitrary continuous flow field. The boat-sail Voronoi diagram is a very natural generalization, but the computation of the associated distance is not easy. This seems the main reason why this Voronoi diagram has not been studied from a computational point of view.

In this paper, we construct a numerical method for computing the boat-sail distances and the associated Voronoi diagram. For this purpose, we first reduce the problem to a boundary value problem of a partial differential equation. A similar idea was already used for the Euclidean distance, that is,

the problem of computing the Euclidean distance was reduced to a boundary value problem of a partial differential equations called the eikonal equation [18]. Hence, our formulation can be considered a generalization of the eikonal equation.

Our partial differential equation is not linear but quadratic, and hence the standard methods such as the finite element method and the finite difference method cannot be used directly, because the resulting system of algebraic equations is not linear. Instead, we use the idea of the fast marching method proposed by Sethian [18], which was originally constructed for the eikonal equation.

In Section 2, we introduce a mathematical model for the boat-sail distance and the associated Voronoi diagram. In Section 3, we derive a partial differential equation for representing the boat-sail distance. In Section 4, we overview the fast marching method and a numerical method based on the fast marching method is constructed for computing this distance and the associated Voronoi diagram. However, this method turns out numerically unstable, and hence, next in Section 5, we construct a new method, which is a combination of the fast marching method and the finite-element method. Finally, we give concluding remarks in Section 6.

2 Boat-Sail Distance and the Associated Voronoi Diagram

Let $\Omega \subset \mathbf{R^2}$ denote a two-dimensional domain with an (x, y) Cartesian coordinate system, and let $f(x, y) \in \mathbf{R^2}$ be a two-dimensional vector given at each point (x, y) in Ω. A physical interpretation is that Ω corresponds to the surface of water and $f(x, y)$ represents the velocity of the water flow. Hence, we call $f(x, y)$ the flow field. We assume that $f(x, y)$ is continuous in Ω.

Consider a boat that has the maximum speed F on the still water, that is, the boat can move at speed F in any direction if there is no flow of water. Let Δt denote a short time interval. Suppose that the driver tries to move the boat at speed F in the direction v_F, where v_F is the unit vector, and hence the boat will move from the current point p to $p + \Delta t F v_F$ in time Δt if there is no water flow, as shown by the broken arrow in Fig 1. However, the flow of water also displaces the boat by $\Delta t f(x, y)$, and hence the actual movement Δu of the boat in time interval Δt is represented by

$$\Delta u = \Delta t F v_F + \Delta t f(x, y). \tag{1}$$

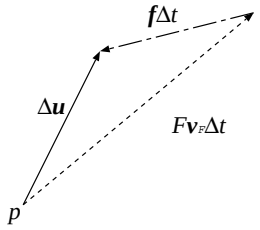


Fig. 1. Relations among the movement Δu, the water flow f and the velocity $F v_F$ of the boat.

Consequently, the effective speed of the boat in the water flow in given by

$$\left| \frac{\Delta u}{\Delta t} \right| = |F v_F + f(x, y)| . \tag{2}$$

We assume that F is large enough to satisfy the condition

$$F > \max_{(x,y)\in\Omega} |f(x, y)|. \tag{3}$$

Hence, the boat can move in any direction against the flow even if the direction of the boat sailing is opposite to the direction of the flow.

Let p and q be two points in Ω, and let $c(s) \in \Omega$ denote a curve from p to q with the arc-length parameter s $(0 \le s \le \overline{s})$ such that $c(0) = p$ and $c(\overline{s}) = q$. Then, the time, say $\delta(c, p, q)$, necessary for the boat to move from p to q along the curve $c(s)$ with the maximum speed is obtained by

$$\delta(c, p, q) \equiv \int_0^{\overline{s}} \left(\frac{1}{\left| \frac{\Delta u}{\Delta t} \right|} \right) \mathrm{d}s = \int_0^{\overline{s}} \frac{1}{|F v_F + f(x, y)|} \mathrm{d}s, \tag{4}$$

where $\equiv$ implies that the symbol in the left-hand side is defined by the expression in the right-hand side.

Let C be the set of all paths from p to q. We define $d(p, q)$ by

$$d(p, q) \equiv \min_{c \in C} \delta(c, p, q). \tag{5}$$

That is, $d(p, q)$ represents the shortest time necessary for the boat to move from p to q. We can consider that $d(p, q)$ is proportional to the effective distance from p to q, and hence we abuse the term "distance" and call $d(p, q)$ the *boat-sail distance* from p to q. Note that $d(p, q)$ is not symmetric; the time necessary to move from p to q is not in general equal to the time necessary to move from q to p.

Next, we define a generalized Voronoi diagram with respect to the boat-sail distance. Let $P = \{p_1, p_2, \cdots, p_n\}$ be a set of n points, called boat harbors, in Ω. For $p_i \in P$, we define region $R(P; p_i)$ by

$$R(P; p_i) \equiv \bigcap_{j \neq i} \{p \in \Omega \mid d(p_i, p) < d(p_j, p)\}. \tag{6}$$

$R(P; p_i)$ represents the set of points which the boat at harbor p_i can reach faster than any other boats. The domain Ω is partitioned into $R(P; p_1)$, $R(P; p_2)$, $\cdots$, $R(P; p_n)$ and their boundaries. This partition is called the *Voronoi diagram for the boat-sail distance* or the *boat-sail Voronoi diagram* for short.

Note that this Voronoi diagram is a generalization of the Voronoi diagram in a river; actually, the Voronoi diagram in a river corresponds to the case where $f(x, y)$ is a constant in Ω.

3 Reduction to a Boundary Value Problem

Suppose that we are given the flow field $f(x, y)$ and the point $p_0 = (x_0, y_0)$ of the boat harbor in Ω. Let $T(x, y)$ be the shortest arrival time at which the boat departing p_0 at time 0 can reach the point $p = (x, y)$, that is, $T(x, y) \equiv d(p_0, p)$.

In this section, we derive the partial differential equation that should be satisfied by the unknown function $T(x, y)$.

Let C be an arbitrary positive constant. The equation $T(x, y) = C$ represents a curve, any point on which can be reached in time C by the boat departing p_0 at time 0. As shown in Fig. 2, assume that the boat moving along a shortest path passes through the point (x, y) at time C and reaches the point $(x + \Delta x, y + \Delta y)$ at time $C + \Delta t$, where Δt is positive and small. Hence, in particular, we get

$$T(x + \Delta x, y + \Delta y) - T(x, y) = \Delta t. \tag{7}$$

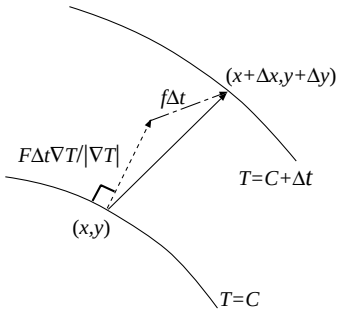


Fig. 2. Decomposition of the movement of a boat.

The motion of the boat from (x, y) to $(x + \Delta x, y + \Delta y)$ can be decomposed into two motion, one is the progress of the boat on virtual still water and the other is the motion due to the water flow.

If there is no flow of water, the shortest path should be perpendicular to the curve $T = C$, and hence, the progress of the boat during time interval Δt is represented by $F \frac{\nabla T}{|\nabla T|} \Delta t$. On the other hand, the displacement of the boat caused by the flow of water is $f \Delta t$. Hence, the total motion of the boat is represented by

$$F \frac{\nabla T}{|\nabla T|} \Delta t + f \Delta t. \tag{8}$$

Let us denote

$$T_x \equiv \frac{\partial T}{\partial x}, \quad T_y \equiv \frac{\partial T}{\partial y}. \tag{9}$$

Also let $g(x, y)$ and $h(x, y)$ denote the first and second components of $f(x, y)$. Then from the equation (8), we get

$$\Delta x = F\frac{T_x}{|\nabla T|}\Delta t + g\Delta t, \quad \Delta y = F\frac{T_y}{|\nabla T|}\Delta t + h\Delta t. \tag{10}$$

Hence, we get

$$\begin{aligned} T(x+\Delta x, y+\Delta y) &= T(x,y) + T_x\Delta x + T_y\Delta y + \mathrm{O}((\Delta x)^2 + (\Delta y)^2) \\ &= T(x,y) + T_x(F\frac{T_x}{|\nabla T|} + g)\Delta t + T_y(F\frac{T_y}{|\nabla T|} + h)\Delta t + \mathrm{O}(\Delta t^2). \end{aligned}$$

Substituting this equation in equation (7), we get

$$T_x(FT_x + g|\nabla T|) + T_y(FT_y + h|\nabla T|) = |\nabla T|, \tag{11}$$

and hence

$$F|\nabla T| = 1 - \nabla T \cdot f. \tag{12}$$

This is the partial differential equation that should be satisfied by the arrival time $T(x, y)$.

In the next section, we consider how to solve this partial differential equation numerically, together with the boundary condition

$$T(x_0, y_0) = 0. \tag{13}$$

4 Numerical Methods Based on the Fast Marching Method

Standard methods for solving boundary value problems are the finite difference method and the finite element method. However, we cannot use these methods for our problem, because our partial differential equation is quadratic, but not linear. On the other hand, our equation has the property that the arrival time $T(x, y)$ is monotone increasing as we move along the shortest paths starting at p_0. A typical equation of this type is the eikonal equation $S|\nabla T| = 1$ [18], where T is the unknown function representing the arrival times, and S is a known speed of evolution that depends on the location (x, y), and the eikonal equation can be solved efficiently and stably by the fast marching method [18]. In this section, we overview the outline of the fast marching method and show the numerical example for our boundary value problem.

4.1 Upwind-like Difference Approximation and the Scheme

In Ω, we place the grid points

$$(x_i, y_j) = (i\Delta x, j\Delta y), \quad i, j = 0, \pm 1, \pm 2, \cdots,$$

where Δx and Δy are small constants and i and j are integers. For each grid point (x_i, y_j), we associate $T_{ij} = T(x_i, y_j)$. $T_{00} = T(x_0, y_0) = 0$ because of the boundary condition (13), while all the other T_{ij}'s are unknown variables.

Starting with the neighbors of (x_0, y_0), we want to compute T_{ij}'s grid by grid from smaller values to larger values. Hence, we use an upwind-like scheme. Let us define the first order forward and backward differences for x and y directions as respectively

$$\begin{aligned} D_{ij}^{+x}T &= \frac{T_{i+1,j} - T_{ij}}{\Delta x}, \quad D_{ij}^{-x}T = \frac{T_{ij} - T_{i-1,j}}{\Delta x}, \\ D_{ij}^{+y}T &= \frac{T_{i,j+1} - T_{ij}}{\Delta y}, \quad D_{ij}^{-y}T = \frac{T_{ij} - T_{i,j-1}}{\Delta y}. \end{aligned}$$

Then, the first order upwind-like operators are written by

$$\begin{aligned} D_{ij}^{x}T &\equiv \max(D_{ij}^{-x}T, 0) + \min(D_{ij}^{+x}T, 0), \\ D_{ij}^{y}T &\equiv \max(D_{ij}^{-y}T, 0) + \min(D_{ij}^{+y}T, 0). \end{aligned}$$

Note that

$$\max(D_{ij}^{-x}T, 0) = \begin{cases} D_{ij}^{-x}T & \text{if } T_{i-1,j} < T_{ij}, \\ 0 & \text{otherwise,} \end{cases} \tag{14}$$

which means that this term represents the left finite difference $D_{ij}^{-x}T$ if and only if the left grid point (x_{i-1}, y_j) is nearer to the boat harbor than the current grid point (x_i, y_i) with respect to the boat-sail distance. The other terms imply similarly. Hence, $D_{ij}^{x}T$ in the equation (14) represents the left finite difference (the first term in the right-hand side of the equation (14)) if the left grid point is nearer to the boat harbor while it represents the right finite difference (the second term in the right-hand side of the equation (14)), if the right grid point is nearest to the harbor. Similarly $D_{ij}^{y}T$ in the equation (14) implies the lower finite difference if the lower neighbor is nearer to the harbor, and the upper finite difference if the upper neighbor is nearer to the harbor.

If the values of the nearest and the second nearest gird points to (x_i, y_i) have already been known, we can similarly define the second order forward and backward differences as

$$\begin{aligned} D_{ij}^{\pm x}T \mp \frac{\Delta x}{2}(D_{ij}^{\pm x})^2 T &= \mp\frac{3T_{i,j} - 4T_{i\pm1,j} + T_{i\pm2,j}}{2}, \\ D_{ij}^{\pm y}T \mp \frac{\Delta y}{2}(D_{ij}^{\pm y})^2 T &= \mp\frac{3T_{i,j} - 4T_{i,j\pm1} + T_{i,j\pm2}}{2}, \end{aligned}$$

and the second order upwind-like operators are also written by

$$D_{ij}^{x}T \equiv \max(D_{ij}^{-x}T + \mathrm{sw}_{ij}^{-x}\frac{\Delta x}{2}(D_{ij}^{-x})^2 T, 0) + \min(D_{ij}^{+x}T + \mathrm{sw}_{ij}^{+x}\frac{\Delta x}{2}(D_{ij}^{+x})^2 T, 0), \tag{15}$$

$$D_{ij}^{y}T \equiv \max(D_{ij}^{-y}T + \mathrm{sw}_{ij}^{-y}\frac{\Delta y}{2}(D_{ij}^{-y})^2 T, 0) + \min(D_{ij}^{+y}T + \mathrm{sw}_{ij}^{+y}\frac{\Delta y}{2}(D_{ij}^{+y})^2 T, 0), \tag{16}$$

where

$$\mathrm{sw}_{ij}^{\pm x} = \begin{cases} 1, & \text{if } T_{i\pm2,j} \text{ and } T_{i\pm1,j} \text{ are known and } T_{i\pm2,j} \le T_{i\pm1,j}, \\ 0, & \text{otherwise,} \end{cases} \tag{17}$$

and similarly, in the case of $\mathrm{sw}_{ij}^{\pm y}$, if $T_{i,j\pm2}$ and $T_{i,j\pm1}$ are known, and $T_{i,j\pm2} \le T_{i,j} \pm 1$, $\mathrm{sw}_{ij}^{\pm y} = 1$; otherwise $\mathrm{sw}_{ij}^{\pm y} = 0$.

Let us define

$$g_{ij} \equiv g(x_i, y_j), \quad h_{ij} \equiv h(x_i, y_j).$$

We replace ∇T by $(D_{ij}^{x}T, D_{ij}^{y}T)$ and f by (g_{ij}, h_{ij}) in our target equation (12), and thus we obtain the second order upwind-like scheme of the equation:

$$F^2\{(D_{ij}^{x}T)^2 + (D_{ij}^{y}T)^2\} = (1 - (D_{ij}^{x}T)g_{ij} + (D_{ij}^{y}T)h_{ij})^2. \tag{18}$$

When we solve this equation, the value of T at the neighbor grid points that are nearer to the harbor have already been given. Hence, this equation is quadratic in the single unknown T_{ij}.

4.2 Algorithms

Now we are ready to describe our numerical method. We solve our boundary value problem by the same strategy as the fast marching method by Sethian [18]. This method is similar to the Dijkstra method for computing all shortest paths from a start vertex in a graph. We consider the grid structure the graph in which the vertices are grid points and the edges connect the four neighbors of each grid point. We start with the boat harbor at which $T_{ij} = 0$, and compute T_{ij}'s one by one from the nearest grid point. The only difference from the Dijkstra method is that the quadratic equation (18) is solved to obtain the value of T_{ij}, while in the Dijkstra method, the length of the shortest path is computed by adding the lengths of the edges on the path.

In the next algorithm, the grid points are classified into three groups: "known" points, "frontier" points and "unknown" points. The "known" points are points at which the values T_{ij} are known. The "frontier" points are points that are not yet known but are the neighbors of the "known" points. The "unknown" points are all the other points. In the algorithm, all the points other than the boat harbor are categorized as "unknown" points, and then changed to "frontier" points and eventually to "known" points.

Algorithm 1 (Boat-sail distance from a single harbor)

Input: flow function $f(x, y)$ in Ω and the boat harbor p_0.

Output: Arrival time T_{ij} at every grid point in Ω.

Procedure:

1. Set $T_{ij} \leftarrow 0$ for the grid point corresponding to the harbor, and $T_{ij} \leftarrow \infty$ for all the other points.
2. Name the grid point p_0 as "frontier", and all the other grid points as "unknown".
3. choose the "frontier" point $p = (x_i, y_i)$ with the smallest value of T_{ij}, and rename it as "known".
4. For all the neighbors of p that are not "known", do 4.1, 4.2 and 4.3.
   - 4.1 If p is "unknown", rename it as "frontier".
   - 4.2 Recompute the value of T_{ij} by solving the equation (18).
   - 4.3 If the recomputed value T_{ij} is smaller than the current value, update the value.
5. If all the grid points are "known", stop. Otherwise go to Step 3.

Let N be the number of the grid points in Ω. Step 1 and 2 of the above algorithm are done in $O(N)$ time. Just as the Dijkstra method, we use a heap data structure to store the "frontier" grid points with the key T_{ij}. Then, the addition of a new grid point to the heap and the deletion of the "frontier" grid point with the smallest T_{ij} can be done in $O(\log N)$ time. Hence, each processing of Steps 3 and 4 is done in $O(\log N)$ time. Since Steps 3, 4 and 5 are repeated N times, the total time complexity of Algorithm 1 is $O(N \log N)$.

Next, suppose that there are n boat harbors, and they are numbered $1, 2, \cdots, n$. Then, we need a slight change of the algorithm. We assign another value S_{ij} to each grid point (x_i, y_i) that represents the nearest harbor number. The values S_{ij}'s specify the Voronoi regions of the boat-sail Voronoi diagram. The only differences from Algorithm 1 are the initial setting of the start points (Steps 1 and 2) and the updating of the harbor numbers S_{ij}'s (Step 4.3). The new algorithm is as follows.

Algorithm 2 (Boat-sail Voronoi diagram)

Input: flow function $f(x, y)$ in Ω and n harbors $q_1, q_2, \cdots, q_n$.

Output: Arrival time T_{ij} and the nearest harbor number S_{ij} at each grid point.

Procedure:

1. For $k = 1, 2, \cdots, n$, set $T_{ij} \leftarrow 0$ and $S_{ij} \leftarrow k$ for harbor q_k, and $T_{ij} \leftarrow \infty$ for all the other points.
2. Name the grid points $q_1, q_2, \cdots, q_n$ as "frontier", and all the other grid points as "unknown".
3. choose the "frontier" point $p = (x_i, y_i)$ with the smallest value of T_{ij}, and rename it as "known".
4. For all the neighbors of p that are not "known", do 4.1, 4.2 and 4.3.
   - 4.1 If p is "unknown", rename it as "frontier".
   - 4.2 Recompute the value of T_{ij} by solving the equation (18).
   - 4.3 If the recomputed value T_{ij} is smaller than the current value, update the value T_{ij} and also update S_{ij} as the harbor number of the neighbor grid points whose values are used in solving the equation (18).
5. If all the grid points are "known", stop. Otherwise go to Step 3.

Since the updating of the harbor number S_{ij} is done in the same time as the updating of the arrival time T_{ij}, the time complexity of Algorithm 2 is the same as that of Algorithm 1, that is, $O(N \log N)$. Note that the time complexity does not depend on the number n of the harbors. This is because the arrival time T_{ij} is computed only for the nearest harbor unless the grid point is near the boundary of the Voronoi diagram.

4.3 Numerical Experiments

Using Algorithm 2, we constructed the Voronoi diagram in the flow field. Here we assumed that the water flows at the same speed uniformly left to right. Fig. 3 shows the Voronoi diagram for the flow of speed 0.4, generated by 10 generators randomly located in a square region.

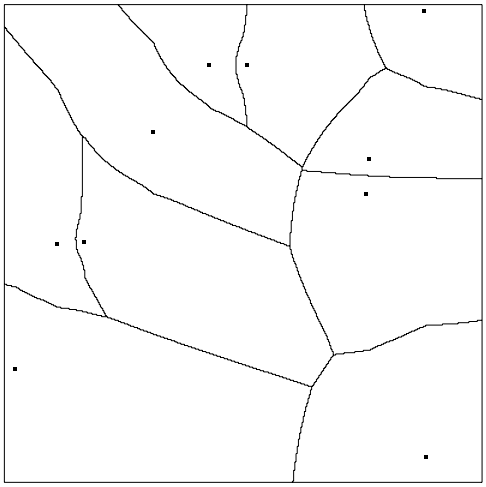


Fig. 3. Voronoi diagram by Algorithm 2 for a constant flow $f = (0.4, 0)$.

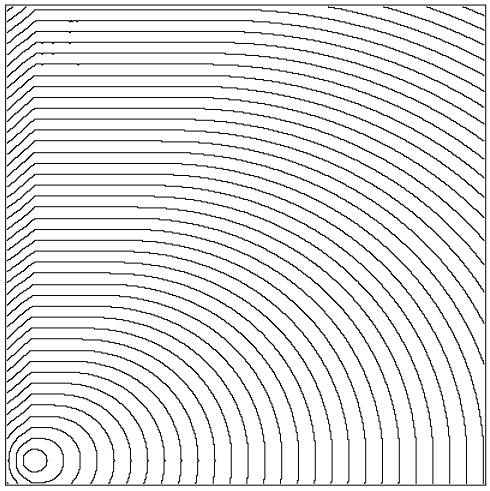


Fig. 4. Isoplethic circle constructed by the fast marching method.

In this case, the bisectors in this Voronoi diagram should essentially be hyperbola [20]. Almost all bisectors, however, are not hyperbola as seen in Fig. 3, that is, the bisectors have some distortions. The distortions arise along the vertical lines extended from the generators. In order to investigate the reason of this phenomenon, we plotted in Fig. 4 the isoplethic curve spreading from a single generator in uniform flow. In this situation, the isoplethic curves must essentially be circles. The computed isotropic curves, however, are distorted in the fan-shape area between the vertical line and the slanted line extending in the upper right direction.

We can consider two reasons for this.

Firstly, as pointed out by Sethian [19], the fast marching method sometimes visits a grid point with larger arrival time before it visits a grid point with smaller arrival time. Let us consider the grid points p_0, p_1 and p_2 in the situation shown in Fig. 5, where p_0 is the start point. The two broken half lines represent the set of points which have the fastest arrival time among the points on each line parallel to the x axis (the slopes of these lines are $\pm \tan |f|$). In this case, the actual arrival time at p_2 is faster than the arrival time at p_1. However, in the fast marching method, p_1 is computed before p_2 is computed. Consequently, the computation for the arrival time at p_2 is carried out using the arrival time at p_1, which is bigger than the true arrival time at p_2. Hence, it is impossible to correctly compute the arrival time at p_2.

Secondly, sometimes the flow is ignored in the computation. Let us look at the grid points $q_1, q_2, \ldots, q_5$ shown in Fig. 5. We consider the arrival time at q_3. When the algorithm computes the arrival time at q_3 for the first time, the equation (18) is solved with $D^x_{ij} = 0$, because only q_1 and q_5 are known. Later the equation (18) is solved again using q_1, q_4 and q_5. Comparing the former arrival time with the latter one, we find that the former is faster than the latter. This is because, in the former case, the water flow is ignored due to $D^x_{ij} = 0$. One strategy to avoid this difficulty might be to inhibit the computation using the data only along one line. However, even in that case, the arrival time at q_3 is computed by the fast marching method with the first-order scheme in the x direction and the second-order scheme in the y direction, because $T(q_4) < T(q_2)$. The first-order scheme in this case implies that the arrival time is linear between q_3 and q_4 although there is a local minimum between them as shown by the broken line in Fig. 5. Consequently, the arrival time is not precise.

We can think that the distortions of the bisectors in Fig. 3 are caused by these two reasons.

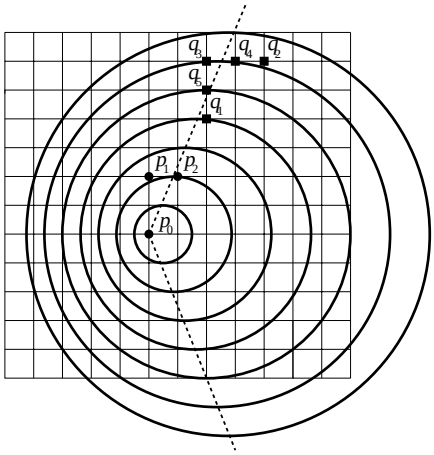


Fig. 5. Example of the situation such that numerical instability occurs.

5 FEM-like Formulation

As we have seen in the previous section, a standard fast marching method does not work for our equation. This is mainly because the arrival time is smaller at the diagonal neighbor than at the vertical and horizontal neighbor. For this kind of a situation, Sethian recommended to use the triangulated version of the fast marching method [18]. However, since his method is a linear approximation on triangular domains, the accuracy is not enough for our purpose. In addition, since the method is a geometric one, it is difficult to apply the method to our partial differential equation. In order to overcome these difficulties, we propose the second order triangulated fast marching method.

5.1 FEM-like Difference Approximation

In the finite element method, the domain $\Omega \subset \mathbf{R}^2$ is divided into many small regions called finite elements, which are triangles or rectangles, and the value of an interior point of a finite element is interpolated from the values on the vertices and on the edges of the finite element. In order to construct a scheme comparable to the second order upwind-like scheme in the fast marching method, we use a quadratic triangular element as follows.

Consider a triangular element shown in Fig. 6(a), where the coordinates of nodes 1, 2 and 3 are (x_1, y_1), (x_2, y_2) and (x_3, y_3), respectively and nodes 4, 5 and 6 are the middle points of the edges. Let $T_1, T_2, \ldots, T_6$ be the values at nodes $1, 2, \ldots, 6$, respectively. Then, the interpolation function T which represents the value at point (x, y) in the triangular element is represented by

$$\begin{aligned} T(x,y) &= T_1\phi_1(x,y)(2\phi_1(x,y)-1) + T_2\phi_2(x,y)(2\phi_2(x,y)-1) + T_3\phi_3(x,y)(2\phi_3(x,y)-1) \\ &\quad +4T_4\phi_2(x,y)\phi_3(x,y) + 4T_5\phi_3(x,y)\phi_1(x,y) + 4T_6\phi_1(x,y)\phi_2(x,y), \end{aligned} \tag{19}$$

where ϕ_1, ϕ_2 and ϕ_3 are the area coordinate functions:

$$\phi_i(x,y) = \frac{1}{D}(a_i + b_i x + c_i y), \tag{20}$$

where

$$D = \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} \quad \text{and} \quad \begin{cases} a_1 = x_2y_3 - x_3y_2, & b_1 = y_2 - y_3, & c_1 = x_3 - x_2, \\ a_2 = x_3y_1 - x_1y_3, & b_2 = y_3 - y_1, & c_2 = x_1 - x_3, \\ a_3 = x_1y_2 - x_2y_1, & b_3 = y_1 - y_2, & c_3 = x_2 - x_1. \end{cases}$$

The functions $\phi_1\phi_2$, $\phi_2\phi_3$, $\phi_3\phi_1$ and $\phi_i(2\phi_i - 1)$ for $i = 1, 2, 3$ are called shape functions.

Next, we consider partial derivatives $\frac{\partial T}{\partial x}$ and $\frac{\partial T}{\partial y}$ of this interpolation function. In this case, we can get the derivatives by differentiating each shape functions in equation (19) by x and y. The derivatives of each shape function are represented by the following: as to $\phi_i(2\phi_i - 1)$ for $i = 1, 2, 3$,

$$\frac{\partial}{\partial x}\phi_i(2\phi_i - 1) = \frac{b_i(4\phi_i - 1)}{D}, \quad \frac{\partial}{\partial y}\phi_i(2\phi_i - 1) = \frac{c_i(4\phi_i - 1)}{D},$$

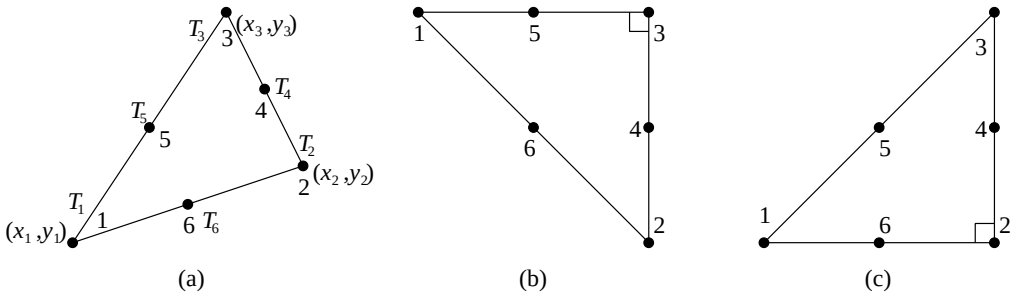


Fig. 6. Examples of triangular finite elements: the number $1, 2, \ldots, 6$ are nodes and $T_1, T_2, \ldots T_6$ are the values at the nodes.

and as to $\phi_1\phi_2$, $\phi_2\phi_3$ and $\phi_3\phi_1$,

$$\frac{\partial}{\partial x}\phi_i\phi_j = \frac{b_i\phi_j + b_j\phi_i}{D}, \quad \frac{\partial}{\partial y}\phi_i\phi_j = \frac{c_i\phi_j + c_j\phi_i}{D}.$$

Hence, the derivatives of T are written by

$$\begin{aligned}\frac{\partial T}{\partial x}(x,y) &= \frac{(b_1T_1 + b_2T_6 + b_3T_5)\phi_1 + (b_1T_6 + b_2T_2 + b_3T_4)\phi_2}{D} \\ &\quad + \frac{(b_1T_5 + b_2T_4 + b_3T_3)\phi_3 - (b_1T_1 + b_2T_2 + b_3T_3)}{D}, \qquad (21) \\ \frac{\partial T}{\partial y}(x,y) &= \frac{(c_1T_1 + c_2T_6 + c_3T_5)\phi_1 + (c_1T_6 + c_2T_2 + c_3T_4)\phi_2}{D} \\ &\quad + \frac{(c_1T_5 + c_2T_4 + c_3T_3)\phi_3 - (c_1T_1 + c_2T_2 + c_3T_3)}{D}. \qquad (22)\end{aligned}$$

In order to know the values of the partial derivatives at the node 3, we substitute (x_3, y_3) into (21) and (22), and get

$$\begin{aligned}\frac{\partial T}{\partial x}(x_3,y_3) &= \frac{3b_3T_3 + 4(b_2T_4 + b_1T_5) - b_1T_1 - b_2T_2}{D}, \qquad (23) \\ \frac{\partial T}{\partial y}(x_3,y_3) &= \frac{3c_3T_3 + 4(c_2T_4 + c_1T_5) - c_1T_1 - c_2T_2}{D}. \qquad (24)\end{aligned}$$

Assume that this quadratic triangular element is the isosceles triangle such that the node 3 forms a right angle, as shown in Fig. 6(b), and that the length of the horizontal and vertical sides adjacent to the node 3 are $2\Delta x$ and $2\Delta y$, respectively. Then, the partial derivatives become

$$\begin{aligned}\frac{\partial T}{\partial x}(x_3,y_3) &= \frac{3T_3 - 4T_5 + T_1}{2\Delta x}, \\ \frac{\partial T}{\partial y}(x_3,y_3) &= \frac{3T_3 - 4T_4 + T_2}{2\Delta y}.\end{aligned}$$

These derivatives coincide with the second order upwind-like difference in the fast marching method. Thus, the equation (23) and (24) are generalizations of upwind-like differences; even if the node 3 is not a right angle, we can compute the value of the finite difference at the node 3. For instance, in the case where the node 2 forms a right angle as shown in Fig. 6 (c), the upwind-like differences are obtained by

$$\begin{aligned}\frac{\partial T}{\partial x}(x_3,y_3) &= \frac{4(T_4 - T_5) - (T_2 - T_1)}{2\Delta x}, \\ \frac{\partial T}{\partial y}(x_3,y_3) &= \frac{3T_3 - 4T_4 + T_2}{2\Delta y}.\end{aligned}$$

We call (23) and (24) the *second order FEM-like differences*.

Needless to say, if $T_1 < T_5$ or $T_2 < T_4$, the second order these differences cannot be used, and so we have to prepare the first order differences. Consider a triangular element whose vertices are nodes 3, 4 and 5, as shown in Fig. 6. Then the interpolation function T^1 is represented by

$$T^1(x,y) = T_3\phi_3(x,y) + T_4\phi_4(x,y) + T_5\phi_5(x,y), \tag{25}$$

where ϕ_3, ϕ_4 and ϕ_5 are the area coordinate functions; we get these functions by replacing nodes 1 and 2 with nodes 5 and 4 in the equation (20), respectively. Considering the differences of the equation (25) in the same manner as the second order difference, we get

$$\frac{\partial T^1}{\partial x}(x_3,y_3) = \frac{b'_3T_3 + b'_4T_4 + b'_5T_5}{D'}, \tag{26}$$

$$\frac{\partial T^1}{\partial y}(x_3,y_3) = \frac{c'_3T_3 + c'_4T_4 + c'_5T_5}{D'}, \tag{27}$$

where

$$D' = \begin{vmatrix} 1 & x_3 & y_3 \\ 1 & x_4 & y_4 \\ 1 & x_5 & y_5 \end{vmatrix} \quad \text{and} \quad \begin{cases} b'_3 = y_5 - y_4, & c'_3 = x_4 - x_5. \\ b'_4 = y_3 - y_5, & c'_4 = x_5 - x_3, \\ b'_5 = y_4 - y_3, & c'_5 = x_3 - x_4. \end{cases}$$

We call the equation (26) and (27) the *first order FEM-like differences*.

From the discussion above, we can obtain the second order FEM-like operator similar to the upwind-like operator as follows.

Let $\triangle_{123}$ be the triangle whose vertices are the nodes 1, 2 and 3, and $\triangle_{543}$ be the triangle whose vertices are the nodes 5, 4 and 3. Since the triangle $\triangle_{123}$ is similar to the triangle $\triangle_{543}$ and, in addition, the length of each edge of $\triangle_{123}$ is twice as that of the corresponding edge of $\triangle_{543}$, we get

$$D' = \frac{D}{4} \quad \text{and} \quad \begin{cases} b'_3 = b_3/2, & c'_3 = c_3/2. \\ b'_4 = b_2/2, & c'_4 = c_2/2, \\ b'_5 = b_1/2, & c'_5 = c_1/2. \end{cases}$$

Hence, the first order FEM-like differences can be replaced by

$$\frac{\partial T^1}{\partial x}(x_3,y_3) = \frac{2b_3T_3 + 2b_2T_4 + 2b_1T_5}{D} \equiv D_1^xT,$$

$$\frac{\partial T^1}{\partial x}(x_3,y_3) = \frac{2c_3T_3 + 2c_2T_4 + 2c_1T_5}{D} \equiv D_1^yT.$$

In addition, let us define $D_2^xT \equiv \frac{\partial T}{\partial x}(x_3,y_3) - D_1^xT$ and $D_2^yT \equiv \frac{\partial T}{\partial y}(x_3,y_3) - D_1^yT$. Then, the second order differences can be decomposed to

$$\frac{\partial T}{\partial x}(x_3,y_3) = D_1^xT + D_2^xT,$$

$$\frac{\partial T}{\partial y}(x_3,y_3) = D_1^yT + D_2^yT.$$

Consequently, we get the *second order FEM-like operator*

$$D^xT \equiv D_1^xT + \mathrm{sw}D_2^xT, \tag{28}$$

$$D^yT \equiv D_1^yT + \mathrm{sw}D_2^yT, \tag{29}$$

where

$$\mathrm{sw} = \begin{cases} 1, & \text{if } T_1,\ T_2,\ T_4 \text{ and } T_5 \text{ are "known"}, T_2 \le T_4 \text{ and } T_1 \le T_5, \\ 0, & \text{otherwise}, \end{cases} \tag{30}$$

and we can compute the equation (18) by applying these operators (28) and (29) to it in place of the upwind-like operators (15) and (16).

However, there are two things we should note. We will explain one notice at the next section. The other notice is as follows.

We can use only "known" grid points in the operator (28) and (29). This means that D_2^xT and D_2^yT are valid only when T_1, T_2, T_4 and T_5 are "known", and D_1^xT and D_1^yT are valid only when T_4 and T_5 are "known". Therefore, if one of T_4 and T_5 is "frontier" or "unknown", the operators (28) and (29) cannot be defined. In what follows, we call the triangle *available* if T_4 and T_5 are "known".

5.2 Choice of a Triangle

In the second-order operator of Sethian's fast marching method, the two vertical neighbors and the two horizontal neighbors are used to compute the value at the target point, as shown in Fig. 7(a), where the target point is represented by the double circles and the used neighbors are represented by dots. In this sence, Sethian's method uses a triangle with a horizontal edge of length $2\Delta x$ and a vertical edge of length $2\Delta y$ meeting at the target point. Fig. 7(a) shows one of the four possible triangles; the other three are obtained when we rotate the triangle in Fig. 7(a) by $\pi/2$, π and $3\pi/2$ around the target point. Thus, in the second order operator of the fast marching method, we can choose one of the four possible triangles according to the direction of the shortest path to the target point.

In our new operator, on the other hand, we can use eight triangles. Fig. 7(b) shows an example, in which a vertical edge of length $2\Delta y$ and a slant edge meeting at the target point; the other seven triangles can be obtained by rotating this triangle by $\pi/2$, π and $3\pi/2$ around the target point, and by mirroring them with respect to the horizontal and the vertical lines passing through the target point. Therefore, we can use one of the eight possible triangles of the type in Fig. 7(b) instead of the four of the type in Fig. 7(a). Thus, we have larger freedom in the choice of a triangle. In what follows, let us call a triangle of the type in Fig. 7(a) a *standard triangle*, and a triangle of the type in Fig. 7(b) a *sharp triangle*.

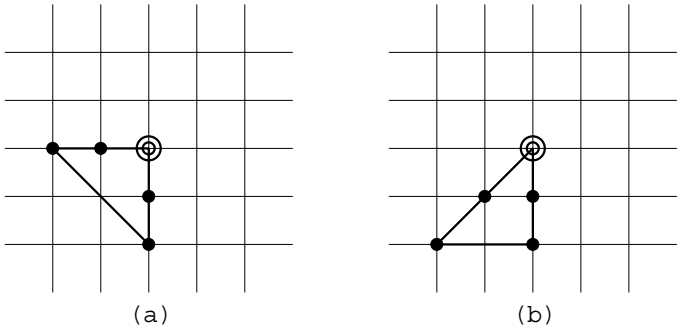


Fig. 7. Freedom in the choice of a triangle.

Consider Fig. 8, which shows the same situation as Fig. 5. Recall that, in this case, the second-order upwind-like operator (i.e., the computation using the standard triangle) cannot precisely compute the arrival time at q_3. On the other hand, in our second-order FEM-like operator, we can use the sharp triangles such as the triangle q_1, q_2, q_3, q_4, q_5 and triangle $q_1, q_2', q_3, q_4', q_5$. Therefore, we can expect that we will obtain a more precise value.

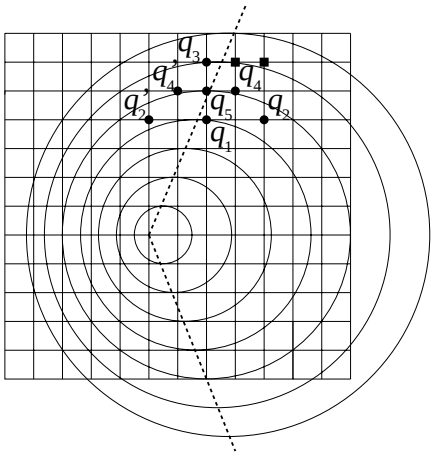


Fig. 8. Example of the case where the second-order upwind-like operator cannot compute, but the second-order FEM-like operator can do.

Now, we can use the eight sharp triangles. The next question is which triangle we should choose for

the most precise computation.

The best triangle is what includes the shortest path to p_3. Consider the triangle p_1, p_2, p_3 shown in Fig. 9. Let n_1 and n_2 be outer normal vectors for the edges p_1p_3 and p_2p_3, respectively. Also, let n_1' (n_2') be the vector directed from $p_1(p_2)$ to p_3. Then, the triangle include the shortest path to p_3 if and only if the direction of the shortest path is between n_1' and n_2'. Hence, from equation (8), this condition can be expressed by

$$\left(F\frac{\nabla T}{|\nabla T|}+f\right)\cdot n_1 \geq 0 \quad \text{and} \quad \left(F\frac{\nabla T}{|\nabla T|}+f\right)\cdot n_2 \geq 0. \tag{31}$$

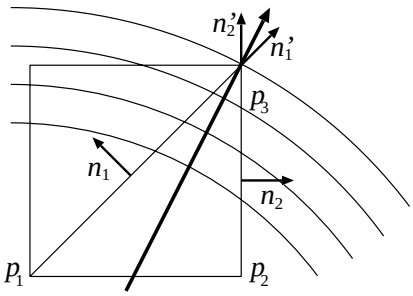


Fig. 9. Relation between the shortest path and the best triangle

Therefore, we have to find the sharp triangle that satisfies the condition (31). However, we have numerical errors in actual computation, and hence cannot expect that the condition (31) is always satisfied strictly. Hence, we choose the sharp triangle that maximizes

$$\min\left\{\left(F\frac{\nabla T}{|\nabla T|}+f\right)\cdot n_1, \left(F\frac{\nabla T}{|\nabla T|}+f\right)\cdot n_2\right\}. \tag{32}$$

Hence, for each target point p, we can construct the following strategy to select the best triangle to compute the value T at p.

Strategy for the Choice of the Best Sharp Triangle

1. Let A be the set of all available sharp triangles for the target point p.
2. For each triangle in A, compute the value T at p using equation (18) with the FEM-like operators (28) and (29), and adopt the value T that maximizes the value (32).

From the discussion above, we get a new scheme by replacing the operators (15) and (16) in the equation (18) by the operators (28) and (29) together with the above strategy. We call this new scheme the *FEM-like scheme*.

Note that this scheme can be applied only when the target point has one or more available sharp triangles. If there is no available triangle at the current target point, we postpone the computation of T until sharp triangles become available.

5.3 Numerical Examples

In this subsection, we show the behavior of our method for computing the boat-sail distances in two problems. One is to obtain the shortest path, and the other is to construct the boat-sail Voronoi diagram.

5.3.1 Shortest Path

Suppose that the flow field f and the boat harbor p is given, and that, for each query point q, we want to find the shortest path from p to q with respect to the boat-sail distance. To solve this problem, we first use Algorithm 1 to compute the arrival time $T(x, y)$ in Ω from the start point p. Next, from each query point p in Ω, we trace back the shortest path, until we reach the starting point p. For this purpose we solve the following ordinary differential equation.

Let $X(t) = (x(t), y(t))$ be the shortest path with parameter t such that

$$X(0) \equiv (x(0), y(0)) = q. \tag{33}$$

We assume that, as t increases, the point $X(t)$ moves along the shortest path in the opposite way from q to p. The motion of the boat in time interval Δt is represented by the expression (8), and hence we get

$$X(t + \Delta t) - X(t) = -\left(F\frac{\nabla T}{|\nabla T|}\Delta t + f\Delta t\right), \tag{34}$$

and consequently we obtain the ordinary differential equation:

$$X_t = -\left(F\frac{\nabla T}{|\nabla T|} + f\right). \tag{35}$$

Thus, the problem of constructing the shortest path is reduced the ordinary differential equation (35) together with the initial condition (33). Since the arrival time $T(x, y)$ at the grid points has been obtained by algorithm 1, the gradient ∇T at any points can be computed by (21) and (22). Hence, we can construct the following algorithm to compute the shortest path.

Algorithm 3 (shortest path)

Input: arrival time $T(x, y)$ from p to all the grid points in Ω and the query point q.

Output: shortest path $X(t)$ from p to q.

Procedure:

1. $X(t) \leftarrow 0$, $t \leftarrow 0$, and fix a small positive real Δt.
2. Find a triangle of the type in Fig. 6(b) or (c) that includes the point $X(t)$, and compute the gradient ∇T at $X(t)$ using (21) and (22).
3. $X(t + \Delta t) \leftarrow X(t) - \left(F\frac{\nabla T}{|\nabla T|}\Delta t + f\Delta t\right)$.
4. $t \leftarrow t + \Delta t$.
5. If $X(t)$ is sufficiently close to p, stop. Otherwise go to Step 2.

Note that we cannot distinguish between the standard triangle and the sharp triangle in Step 2 of the above algorithm, because the point $X(t)$ is not a grid point: we can use any triangle that include $X(t)$.

We show four examples of the shortest paths in the flow field in Figs. 10(a) to 10(d). Here, we assumed that the speed F of a boat be 1. The arrows in the figures represent the directions and the relative speeds of the flow in the field. The lengths of the arrows in the same figure express the relative speeds of the flow; the longer is the arrow, the faster is the flow. The thin curves express the isoplethic curves of the first arrival time, and the thick curve expresses the shortest path.

Fig. 10(a) shows the isoplethic curves in homogeneous flow $f = (0.4, 0.0)$ in a square region. We can see that, the computed value in Fig. 10(a) are much more precise than in Fig. 4. Fig. 10(b) is for the circular flow $f = (-0.7\sin\theta, 0.7\cos\theta)$ in a doughnut region $\{(x, y) \mid 0.25 < x^2 + y^2 < 1\}$. Fig. 10(c) is for the flow field $f = (0.7(1 - y^2), 0.0)$ in a rectangular region $\{(x, y) \mid -1 < y < 1\}$. Finally, Fig. 10(d) shows the case for the flow $f = 0.35(1 - 0.25/z^2), z \in \mathbf{C}$ in the square region $\{(x, y) \mid -1 < |z| < 1\}$. This flow coincides with the theoretical flow pattern obtained when the cylinder of the radius 0.5 is placed at the center of region in the homogeneous, incompressible and nonviscous flow from left to right.

5.3.2 Voronoi Diagram

Here, we show three examples of the Voronoi diagrams in the flow field computed by Algorithm 2.

The first example (Fig. 11(a)) is the Voronoi diagram in the homogeneous flow $f = (0.7, 0)$, generated by 10 generators. The second example (Fig. 11(b)) is the Voronoi diagram in the circular flow $f = (-0.7\sin\theta, 0.7\cos\theta)$ in a doughnut region $\{(x, y) \mid 0.25 < x^2 + y^2 < 1\}$ generated by 10 generators. The last example (Fig. 11(c)) is the Voronoi diagram in the flow field $f = (0.7(1 - y^2), 0)$ in a rectangular region $\{(x, y) \mid -1 < y < 1\}$ generated by 10 generators.

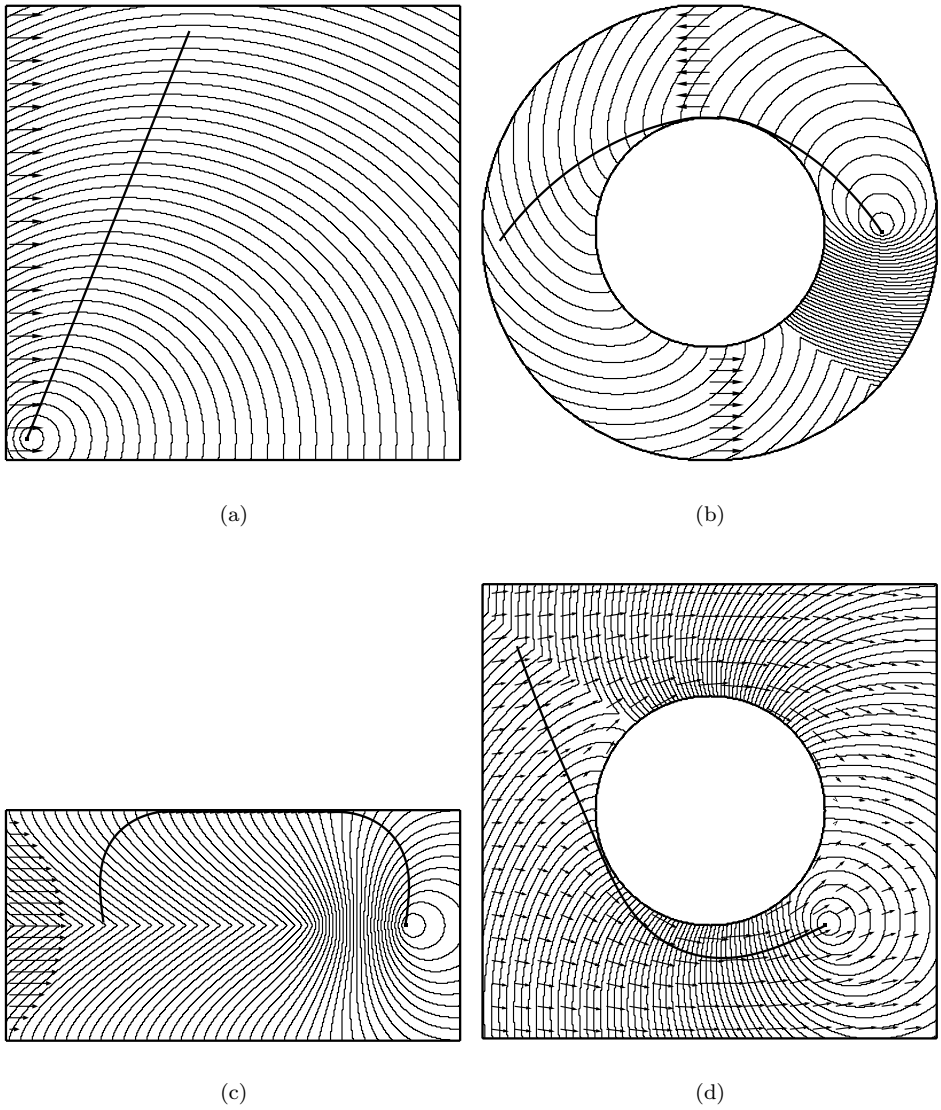
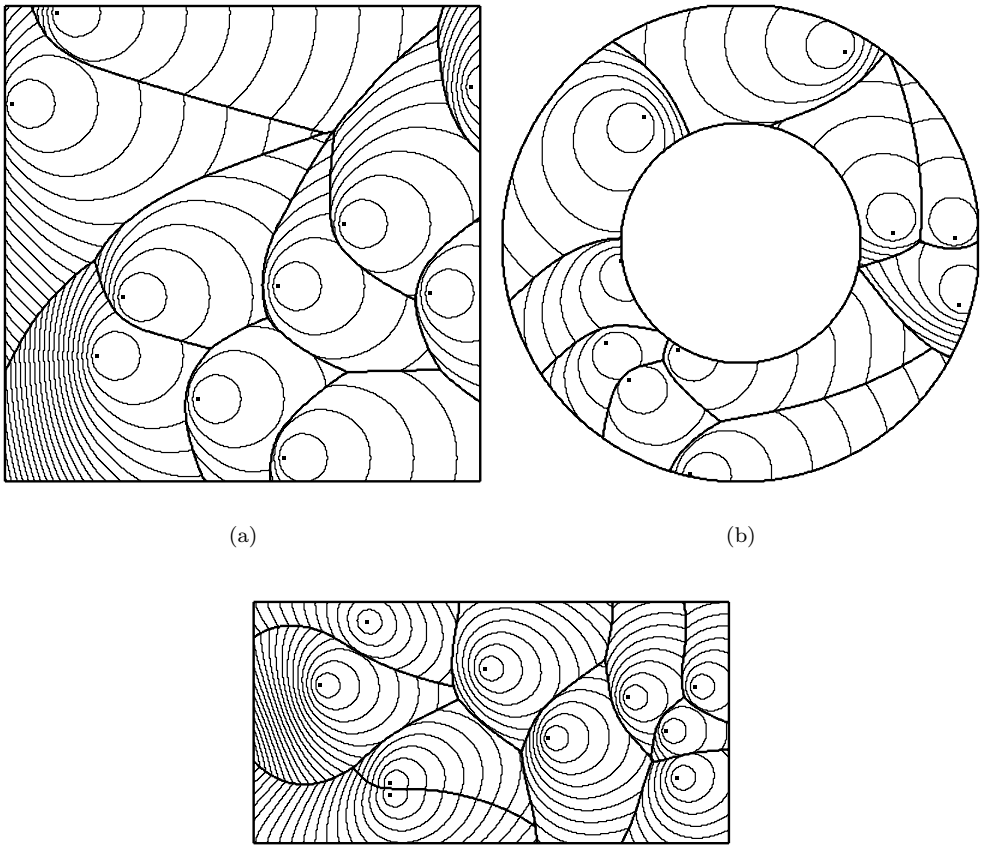


Fig. 10. Isoplethic curves of the first arrival time and the shortest paths to query points.

6 Concluding Remarks

We first defined the boat-sail distance and the associated Voronoi diagram, next derived the partial differential equation satisfied by the first arrival time, thirdly proposed a new scheme for computing the distances and the Voronoi diagrams, and finally showed computational experiments. The concept of the boat-sail distance is natural and intuitive, but the computation is not trivial. Actually the original definition of the boat-sail distance given by the equations (4) and (5) does not imply any explicit idea for computing this distance, because the shortest path is unknown. This seems the main reason why these concepts have not been studied from the computational point of view.

Our breakthrough toward efficient computation is that we succeeded in formulating the problem as the boundary value problem. The distance is defined according to the notion of the boat sailing, and hence a naive formulation will reach an initial value problem of a partial differential equation containing the time variable and its derivatives. In this paper, on the other hand, we concentrated on the first arrival time as the unknown function, and thus constructed an equation without time variable. Moreover, this partial differential equation is quadratic, which is not so simple as linear, but is still tractable. This formulation enables us to use the same idea as the fast marching method, which was originally proposed



(a) (b)

(c)

Fig. 11. Voronoi diagrams in the flow fields.

for the eikonal equation, and thus could construct efficient algorithms.

Our formulation can be extended to a three-dimensional space immediately. An example is a submarine sailing through ocean water. The specific gravity of a submarine is almost 1, and hence we can assume that the submarine can sail in any direction in the three-dimensional space at the same maximum speed F if there is no flow of water. Suppose that the water flow field is given by $f(x, y, z)$. Then, the first arrival time $T(x, y, z)$ of the submarine obeys the three-dimensional version of the equation (12). That is, we obtain the three-dimensional version from the equation (12), just by replacing $T(x, y)$ and $f(x, y)$ with $T(x, y, z)$ and $f(x, y, z)$, respectively. Also, algorithm for solving this equation can be obtained from Algorithm 1 directly.

Acknowledgment

This work is supported by the 21st Century COE Program of the Information Science and Technology Strategic Core, and the Grant-in-aid for Scientific Research (S)15100001 of the Ministry of Education, Science, Sports and Culture of Japan.

References

[1] O. Aichholzer, F. Aurenhammer, D. Z. Chen and D. T. Lee: Skew Voronoi diagrams. *International Journal of Computational Geometry and Applications*, vol. 9 (1999), pp. 235–247.

[2] B. Aronov and J. O'Rourke: Nonoverlap of the star unfolding. *Discrete and Computational Geometry*, vol. 8 (1992), pp. 219–250.

[3] B. Aronov: On the geodesic Voronoi diagram of point sites in a simple polygon. *Algorithmica*, vol. 4 (1989), pp. 109–140.

[4] P. F. Ash and E. D. Bolker: Generalized Dirichlet tessellations. *Geometriae Dedicata*, vol. 20 (1986), pp. 209–243.

[5] F. Aurenhammer: Power diagrams — Properties, algorithms and applications. *SIAM Journal of Computing*, vol. 16 (1987), pp. 78–96.

[6] F. Aurenhammer: Voronoi diagrams — A survey of a fundamental geometric data structure. *ACM Computing Surveys*, vol. 23 (1991), pp. 345–405.

[7] F. Aurenhammer and H. Edelsbrunner: An optimal algorithm for constructing the weighted Voronoi diagram in the plane. *Pattern Recognition*, vol. 17 (1984), pp. 251–257.

[8] L. P. Chew and R. Drysdale, III: Voronoi diagrams based on convex distance functions. *Proceedings of the ACM Symposium on Computational Geometry*, 1985, Baltimore, pp. 235–244.

[9] S. Fortune: Voronoi diagrams and Delaunay triangulations. In D.-Z. Du and F. K. Hwang (eds.): *Computing in Euclidean Geometry*, World Scientific Publishing, Singapore, 1992, pp. 193–233.

[10] H. Imai, M. Iri and K. Murota: Voronoi diagram in the Laguerre geometry and its applications. *SIAM Journal of Computing*, vol. 14 (1985), pp. 93–105.

[11] R. Klein: Abstract Voronoi diagrams and their applications. *Lecture Notes in Computer Science*, 333 (International Workshop on Computational Geometry, Wurzburg, 1988), Springer-Verlag, Berlin, 1988, pp. 148–157.

[12] K. Kobayashi and K. Sugihara: Crystal Voronoi diagram and its applications. *Future Generation Computer System*, vol. 18 (2002), pp. 681–692.

[13] D.-T. Lee: Two-dimensional Voronoi diagrams in the L_p-metric. *Journal of the ACM*, vol. 27 (1980), pp. 604–618.

[14] R. E. Miles: Random points, sets and tessellations on the surface. *The Indian Journal of Statistics*, Series A, vol. 33 (1971), pp. 145–174.

[15] A. Okabe, B. Boots, K. Sugihara and S. N. Chiu: *Spatial Tessellations — Concepts and Applications of Voronoi Diagrams, Second Edition*. John Wiley and Sons, Chichester, 2000.

[16] B. F. Schaudt and R. L. Drysdale: Multiplicatively weighted crystal growth Voronoi diagrams. *Proceedings of the Seventh Annual ACM Symposium on Computational Geometry*, 1991, pp. 214–223.

[17] T. H. Scheike: Anisotropic growth of Voronoi cells. *Advances in Applied Probability*, vol. 26 (1994), pp. 43–53.

[18] J. A. Sethian: Fast marching method. *SIAM Review*, vol. 41 (1999), pp. 199–235.

[19] J. A. Sethian: *Level Set Methods and Fast Marching Methods, Second Edition*. Cambridge University Press, Cambridge, 1999.

[20] K. Sugihara: Voronoi diagrams in a river. *International Journal of Computational Geometry and Applications*, vol. 2 (1992), pp. 29–48.