Spatial Analysis along Networks

Statistical and Computational Methods

Atsuyuki Okabe

School of Cultural and Creative Studies
Aoyama Gakuin University
Emeritus Professor, University of Tokyo, Japan

Kokichi Sugihara

Graduate School of Advanced Mathematical Sciences
Meiji University
Emeritus Professor, University of Tokyo, Japan

WILEY

A John Wiley & Sons, Ltd., Publication

pointers to those links. In addition, the inter-layer links should contain attributes showing the kinds of connections such as elevators, escalators and staircases.

3.2.2 General nonplanar networks

Many nonplanar networks are difficult to divide into layers in a natural manner. For example, consider international airplane routes connecting airports (Figure 3.10). If we draw the routes by arcs on a plane, they intersect each other at many points. However, it is difficult to divide the routes into layers. Actually, each airplane should keep to one of the prespecified discrete heights depending on the direction of flight in order to avoid collision, but even within each height, the airplane routes intersect, and hence the height is different from the 'layer' in which a planar network is embedded. Consequently, we cannot apply the winged-edge data structure to general nonplanar networks.

In order to represent general nonplanar networks, we cannot use 'regions,' and hence the incidence relations between links and nodes should be represented directly in the following manner. At each node $v \in V$, we assign an arbitrary cyclic order to the links incident to v, and regard this cyclic order as *virtual clockwise order* ('virtual' in the sense that it does not necessarily correspond to physical clockwise). For each link l, we virtually use the pointers 'start-c-link' and 'end-c-link' to represent the next links around the start node and the end node, respectively, and also the pointers 'start-cc-link' and 'end-cc-link' to represent the next links in the reverse order around the start node and the end node, respectively. We also use the virtual pointers 'start-node,' 'end-node' and 'node-to-link' in the same manner as in the winged-edge data structure. In other words, we delete from the winged-edge data structure the three pointers 'right-region,' 'left-region' and 'region-to-link,' and use the remaining seven pointers (i.e., 'start-node,' 'end-node,' 'start-c-link,' 'end-c-link,' 'start-cc-link,' 'end-cc-link' and 'node-to-link') to represent nonplanar networks. This data structure enables us to list neighboring elements efficiently in the same manner as the winged-edge data structure. For example, for node $v \in V$, we can list all the links incident to v by Algorithm 3.1 in Section 3.1.3.

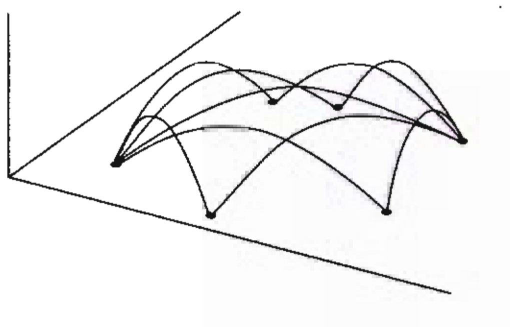

Figure 3.10 Airplane routes.

3.3 Basic geometric computations

Assuming the data structures of planar and nonplanar networks in the preceding section, in this section, we briefly review the methods for representing basic geometric objects such as line segments and polygons, those for computing basic geometric values such as the area of a polygon, and those for geometric tests such as the point-polygon inclusion test. For more details, refer to standard textbooks on computational geometry, such as Preparata and Shamos (1985), Edelsbrunner (1987) and Berg *et al.* (2008).

3.3.1 Computational methods for line segments

A line segment can be represented by the two endpoints. Let v_1 and v_2 be the two endpoints of a line segment in a plane. Then the line segment connecting them is represented by the pair (v_1, v_2) of points, and each point is represented by its coordinates. We denote the coordinates of a node v_i by (x_i, y_i) for any positive integer i throughout this book. An (x, y) coordinate system is said to be a *counterclockwise coordinate system* if the counterclockwise rotation of the x axis by 90 degrees results in the y axis. Throughout this book we assume that the (x, y) coordinate system is counterclockwise unless otherwise mentioned.

3.3.1.1 Right-turn test

Suppose that we are given three points v_1, v_2 and v_3, and that we visit these points one after another in this order. We want to judge whether we turn to the right at v_2. For this purpose we can use the function:

$$F(v_1, v_2, v) = \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x & y \end{vmatrix} = (y_1 - y_2)(x - x_1) - (x_1 - x_2)(y - y_1), \qquad (3.2)$$

where (x, y) represents the coordinates of an arbitrary point v. The equation $F(v_1, v_2, v) = 0$ represents the line passing through v_1 and v_2. We can confirm this as follows. Substituting x_1 and y_1 in x and y, respectively in equation (3.2), we obtain $F(v_1, v_2, v_1) = 0$, which implies that the line passes through v_1. Similarly, substituting x_2 and y_2 in x and y, respectively in equation (3.2), we obtain $F(v_1, v_2, v_2) = 0$, which implies that the line passes through v_2. Therefore, this line passes through both v_1 and v_2.

The function $F(v_1, v_2, v)$ is continuous in v, and it becomes 0 when v is on the line. Therefore, $F(v_1, v_2, v)$ is positive if v is on one side of the line, and is negative on the other side. Using this property, we can judge whether we turn to the right at v_2 from the sign of $F(v_1, v_2, v_3)$. The sign depends on the orientation of the coordinate system. Because we adopt the counterclockwise coordinate system, $F(v_1, v_2, v_3) > 0$ implies a left turn and $F(v_1, v_2, v_3) < 0$ implies a right turn.

3.3.1.2 Intersection test for two line segments

Suppose that we are given two line segments l and l', l connects v_1 and v_2, and l' connects v_3 and v_4. Then v_3 and v_4 are on mutually opposite sides of the line l if and only if:

$$\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} \cdot \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_4 & y_4 \end{vmatrix} < 0. \tag{3.3}$$

Similarly, v_1 and v_2 are on mutually opposite sides of the line l' if and only if:

$$\begin{vmatrix} 1 & x_3 & y_3 \\ 1 & x_4 & y_4 \\ 1 & x_1 & y_1 \end{vmatrix} \cdot \begin{vmatrix} 1 & x_3 & y_3 \\ 1 & x_4 & y_4 \\ 1 & x_2 & y_2 \end{vmatrix} < 0. \tag{3.4}$$

Therefore, l and l' intersect at their interior point if and only if both in equations (3.3) and (3.4) are satisfied.

3.3.1.3 Enumeration of line segment intersections

Given a finite set of line segments, we want to enumerate all the intersection points. An efficient algorithm called the *plane sweep method* was proposed by Bentley and Ottmann (1979). In this method, as shown in Figure 3.11, the plane in which the line segments are placed is swept by a vertical sweepline l_s (the bold line segment) from left to right, with a list, denoted by A, that manages the intersection between the line segments and the sweepline in the following manner. As the sweepline moves from left to right, three kinds of events happen. The first kind of an event is that the sweepline hits the left endpoint of a line segment (e.g., at $x_1, x_2, x_3, x_4, x_{10}, x_{11}$ in Figure 3.11). In this case, the new line segment is inserted in the list A. The second kind of an event is that the sweepline hits the right endpoint of a line segment (e.g., at $x_6, x_7, x_8, x_9, x_{12}, x_{13}$ in Figure 3.11). In this case, the line segment is deleted from the list A. The third kind of an event is that the sweepline hits a point of intersection of two line segments (e.g., at x_5 in Figure 3.11). In this case, the two line segments

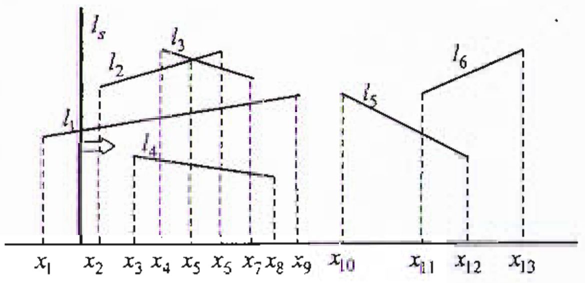

Figure 3.11 Plane sweep method for the enumeration of intersections between line segments.

are interchanged in the list A. As the sweepline l_s moves, the list changes, for instance, in Figure 3.11: $A(x_1) = (l_1)$, $A(x_2) = (l_1, l_2)$, $A(x_3) = (l_4, l_1, l_2)$, $A(x_4) = (l_4, l_1, l_2, l_3)$, $A(x_5) = (l_4, l_1, l_3, l_2)$, $A(x_6) = (l_4, l_1, l_3)$, $A(x_7) = (l_4, l_1)$, $A(x_8) = (l_1)$, $A(x_9) = (\emptyset)$, $A(x_{10}) = (l_5)$, $A(x_{11}) = (l_5, l_6)$, $A(x_{12}) = (l_6)$, $A(x_{13}) = (\emptyset)$, where $A(x_i)$ is the list A when the sweepline l_s is at x_i. For pairs of line segments that become neighbors in the list A, we check whether they intersect, and report the intersection if they do. In the case shown in Figure 3.11, l_2 and l_4 never become neighbors of each other in the list A, and nor do l_3 and l_4, and consequently we can skip the intersection check for these pairs of line segments. Moreover, as l_1 and l_5 in Figure 3.11, two line segments whose projections on the x axis do not overlap (such a pair will never intersect) do not appear in the list A simultaneously, and hence we can similarly skip the intersection check. Thus we can expect to save computational cost if the number of intersections is small. This method can enumerate all the intersections efficiently. In order to evaluate the efficiency of this algorithm, we need a new concept called *time complexity*, which is defined in the next subsection.

3.3.2 Time complexity as a measure of efficiency

Because a network dataset is usually very large, computational methods should be efficient. In this subsection, we show how to measure the efficiency of a computational algorithm. Let $f(n)$ and $g(n)$ be two positive-value functions of n. If there exists a constant C satisfying:

$$\frac{f(n)}{g(n)} \leq C$$

for any n, we write:

$$f(n) = O(g(n))$$

and say that $f(n)$ is of *order* $g(n)$. The fact that $f(n) = O(g(n))$ implies that the rate of increase of $f(n)$ is no faster than that of $g(n)$, and hence $g(n)$ is regarded as the upper bound on the growth of $f(n)$. In particular, if $f(n)$ represents the number of steps necessary for executing an algorithm, we choose a simple function $g(n)$ satisfying $f(n) = O(g(n))$, and regard $g(n)$ as a measure for evaluating the efficiency of the algorithm; we call this the *time complexity* of the algorithm. Specifically, if $f(n)$ represents the average number of steps, the associated $O(g(n))$ is called the *average time complexity*, while if $f(n)$ represents the worst-case number of steps, $O(f(n))$ is called the *worst-case time complexity* (Aho, Hopcroft and Ullman, 1974; Cormen *et al.*, 2001).

Using this measure of time complexity, we can see that the plane sweep method can enumerate all the intersections in $O((n + k)\log n)$ time steps, where n is the number of line segments and k is the number of intersections (Bentley and Ottmann, 1979). Chazelle and Edelsbrunner (1992) improved the time complexity to $O(n \log n + k)$ by constructing a different algorithm, and Balaban (1995) further improved the algorithm by reducing required memory size.

3.3.3 Computational methods for polygons

Let Π be a polygon whose vertices are $p_1, p_2, \ldots, p_n$, located counterclockwise on the boundary of Π. We represent this polygon by $\Pi = (p_1, p_2, \ldots, p_n)$. Because there is freedom in the choice of the start vertex of this list, this polygon is equivalent to $\Pi = (p_i, p_{i+1}, \ldots, p_n, p_1, \ldots, p_{i-1})$ for any i $(1 \le i \le n)$.

3.3.3.1 Area of a polygon

The area A of polygon $\Pi = (p_1, p_2, \ldots, p_n)$ can be computed by:

$$A = \frac{1}{2}\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i), \tag{3.5}$$

where (x_i, y_i) is the coordinates of x_i (x_{n+1} and y_{n+1} should be read as x_1 and y_1, respectively). This formula can be regarded as the summation of the following signed areas of the triangles forming the polygon Π. First, note that:

$$A_i = \frac{1}{2}(x_i y_{i+1} - x_{i+1} y_i)$$

represents the signed area of the triangle formed by the vertices p_i, p_{i+1} and the origin of the coordinate system (Figure 3.12a), where $A_i > 0$ if the origin is to the left of the directed line from p_i to p_{i+1}, $A_i < 0$ if the origin is to the right, and $A_i = 0$ if the origin is on the line passing through p_i and P_{i-1} (Section 3.3.1.1). For example, for the polygon in Figure 3.12a, the signed area A_1 for the triangle Op_1p_2 is positive and similarly the areas A_2, A_3, A_4 are positive (Figure 3.12b), while the area A_5 for the triangle Op_5p_6 and the area A_6 for the triangle Op_6p_1 are negative (Figure 3.12c). Therefore, the summation of these signed areas cancels the part of the triangles outside Π and consequently results in the net area of Π.

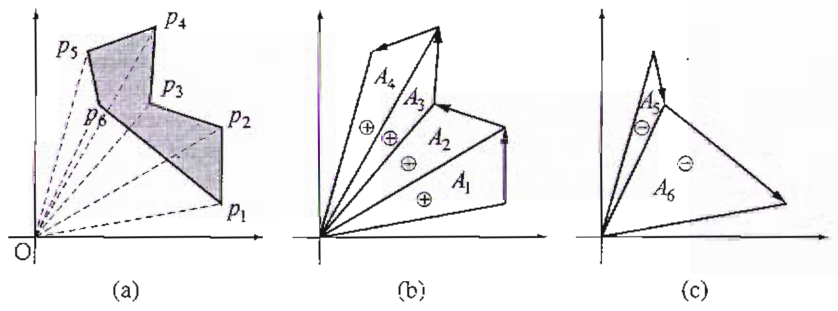

Figure 3.12 Computation of the area of a polygon: (a) a polygon, (b) positively signed areas, (c) negatively signed areas.

3.3.3.2 Center of gravity of a polygon

The center of gravity of a triangle, (x_g, y_g), with vertices v_1, v_2 and v_3 can be computed by:

$$(x_g, y_g) = \left(\frac{x_1 + x_2 + x_3}{3}, \frac{y_1 + y_2 + y_3}{3}\right). \tag{3.6}$$

To compute the center of gravity, (x_g, y_g), of a polygon Π, we first partition the polygon into triangles, $T, i = 1, \ldots, n_i$, next compute the centers of gravity, $(x_{gi}, y_{gi}), i = 1, \ldots n$, of these triangles, and finally compute the weighted average of these centers of gravity with the areas $|T|, i = 1, \ldots, n_i$ chosen as the weights, i.e.:

$$(x_g, y_g) = \left(\sum_{i=1}^{n}\frac{|T_i|}{\sum_{j=1}^{n}|T_j|}x_{gi}, \sum_{i=1}^{n}\frac{|T_i|}{\sum_{j=1}^{n}|T_j|}y_{gi}\right). \tag{3.7}$$

3.3.3.3 Inclusiou test of a point with respect to a polygon

To test whether a point p is included in a polygon Π, we first generate a half line starting at p in an arbitrary direction, and next count the number of intersections between the half line and the boundary of Π. Then, we can judge that p is in Π if the number of intersections is odd as in Figure 3.13a, while we judge p is outside Π if the number of intersections is even as in Figure 3.13b.

Note that this procedure is not perfect because we often come across exceptional cases. An example of an exception is shown in Figure 3.13c, where the half line emanating from p includes a complete link of Π. Exceptional cases in geometric configurations are said to be *degenerate*. In general, there are many different degenerate situations, and it is usually difficult to give a complete algorithm that can cope with all the degenerate cases. However, as mentioned in Section 3.1.5, there is a powerful technique, called *symbolic perturbation*, which can remove degenerate cases. With this method, we can construct algorithms without worrying about exceptional branches for degenerate cases. For details, see Edelsbrunner and Mucke (1988) and Yap (1988).

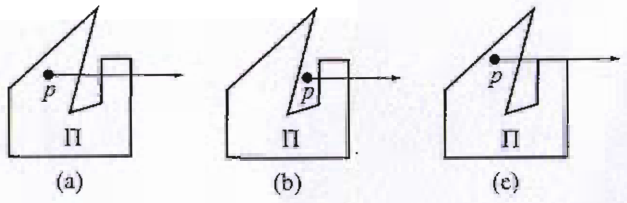

Figure 3.13 Inclusion test of a point with respect to a polygon: (a) a point (the black circle) included in a polygon, (b) a point outside a polygon, (c) a degenerate case.

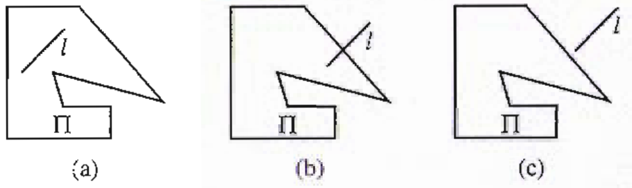

Figure 3.14 Possible relations between a polygon and a line segment: (a) a line segment l included in a polygon, (b) a line segment intersected a polygon, (c) a line segment outside a polygon.

3.3.3.4 Polygon-line intersection

Consider the intersection between a polygon Π and a line segment l (e.g., Figure 3.14). The relative relationship between Π and l has one of the following three possibilities:

(i) l is completely in Π as in Figure 3.14a,
(ii) l intersects with the boundary of Π as in Figure 3.14b, or
(iii) l is completely outside Π as in Figure 3.14c.

We can identify these possibilities by the following algorithm.

Algorithm 3.3 (Point-line intersection)

Input: polygon Π and line segment l.
Output: one of the possibilities (i), (ii), (iii) of the intersection.
Procedure:
1. Test whether l intersects one of the edges of Π. If it intersects, report (ii).
2. Otherwise, test whether one endpoint of l is included in Π. If it is, report (i).
3. Otherwise, report (iii).

Note that in Steps 1 and 2, the intersection test (Section 3.3.1.2) and the inclusion test (Section 3.3.3.3) are used, respectively.

3.3.3.5 Polygon intersection test

Consider the intersection between two polygons Π_1 and Π_2. As shown in Figure 3.15, the relationship between Π_1 and Π_2 has one of the following four possibilities:

(i) the boundary of Π_1 intersects that of Π_2 (Figure 3.15a),
(ii) Π_1 is completely included in Π_2 (Figure 3.15b),
(iii) Π_2 is completely included in Π_1 (Figure 3.15c), or
(iv) Π_1 and Π_2 do not overlap at all (Figure 3.15d).

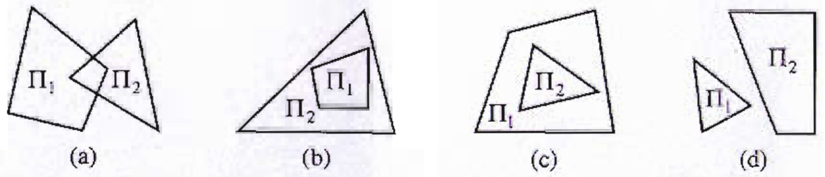

Figure 3.15 Possible relations between two polygons: (a) intersected polygons, (b) polygon Π_1 completely included in polygon Π_2, (c) polygon Π_2 completely included in polygon Π_1, (d) nonintersected polygons.

We can identify these possibilities by the next algorithm.

Algorithm 3.4 (Polygon intersection)

Input: two polygons Π_1 and Π_2.
Output: one of the possibilities (i), (ii), (iii), (iv) of the intersection.
Procedure:
1. Test whether an edge of Π_1 and an edge of Π_2 intersect for all pairs of edges. If there exists a pair that intersects, report (i).
2. Otherwise, choose a vertex of Π_1 and test whether the vertex is in Π_2.
3. If it is, report (ii).
4. Otherwise, choose a vertex of Π_2, and test whether the vertex is in Π_1. If it is, report (iii).
5. Otherwise, report (iv).

Again the intersection test and the inclusion test are used in this algorithm.

3.3.3.6 Extraction of a subnetwork inside a polygon

We consider the problem of extracting the part of a network that is included in a given polygon, as shown in Figure 3.16. Let N be a planar connected network and Π be a polygon, and we want to extract the subnetwork of N that is included in Π. This problem can be solved in the following manner.

First we overlay N and Π and find all the points of intersections between the links of N and the edges of Π by the plane sweep method (Section 3.3.1.3). Then we have two possible cases, Case 1: no points of intersection are found (Figure 3.16a and b), or Case 2: one or more points of intersections are found (Figure 3.16c).

Case 1. Suppose that we find no point of intersection between N and Π. Then, we arbitrarily choose one node of N (the black circles in Figure 3.16a and b) and execute the inclusion test of the node with respect to Π (Section 3.3.3.3). If it is included, we report that the whole network N is included in Π (Figure 3.16a). Otherwise, we report that no part of N is included in Π (Figure 3.16b).