Chapter 7
Spatial Operation

7.1 Introduction

Q: What is spatial operation?
A: Spatial operation is computational manipulation of spatial objects that deepen our understanding of spatial phenomena. In spatial analysis, we perform spatial operation after visual analysis.

Spatial operation includes the following contents.
- Spatial overlay
- Spatial search
- Buffer operation
- Voronoi diagram
- Delaunay triangulation
- Network operation

Individual spatial operations are simple and easy to do in GIS. However, a combination of spatial operations is a powerful tool for analyzing spatial phenomena.

Spatial operation are based on computational algorithms developed in a research field called ‘computational geometry.’ Computational geometry is a subfield of computer science, which was first advocated by Ian Shamos in 1986 in his dissertation.

Since then, new computational (geometrical) algorithms have been developed, which gives a technical foundation of today’s GIS.

• References - computational geometry

Spatial overlay is a spatial operation that puts a map layer on another layer to produce a new layer.

This operation is different from ‘visual overlay.’ Visual overlay displays two layers simultaneously on one device, but it does not create new spatial data.

Spatial overlay reconstructs the topology of spatial objects when they are represented by arc-node structure.

ArcInfo has six types of spatial overlay operations.

1. Intersect
2. Union
3. Identity
4. Clip
5. Erase
6. Update (keepborder, dropborder)

The six overlay operations produce different outputs from the same input.

Intersect and Union correspond to the operators ‘AND’ and ‘OR’ in formal logic.

Both Intersect and Union are symmetric in the sense that the order of two layers does not affect the output.

Identity, on the other hand, is not symmetric. The first layer serves as the base layer while the second layer is the overlaid layer. We have a different result if we change the order of two layers.
7.2.3 **Clip and Erase**

Clip and Erase extract spatial objects from a map layer by using another layer (clip layer).

Clip layer works as a cutting tool for extracting spatial objects. Clip layer specifies the area on the input layer from which spatial objects are extracted.

Clip and Erase are asymmetric operations; the order of layers affects the output.

---

7.2.4 **Update**

Update literally ‘updates’ a part of (or the whole) map layer by another layer.

Update has an option ‘keepborder’ or ‘dropborder’. If we choose ‘keepborder’, Update keeps the boundaries between input and new layers as they are. ‘Dropborder’ removes the boundaries between the two layers after updating the input layer.

---

In overlay operations, the attribute data of spatial objects in an input layer are usually transferred to the output layer.
7.2.5 Basic calculations used in spatial overlay

Spatial overlay frequently calculates intersections of two line segments and the area of polygons. We thus need efficient and accurate algorithms for those calculations.

Calculation of line intersections

Suppose there are two line segments.

The line segments are mathematically represented by:

\[ l_1 : \frac{x - x_1}{x_3 - x_1} + \frac{y - y_1}{y_3 - y_1} = 0 \leq \lambda \leq 1 \]

\[ l_2 : \frac{x - x_2}{x_4 - x_2} + \frac{y - y_2}{y_4 - y_2} = 0 \leq \mu \leq 1 \]

To calculate the intersection of the line segments, we use three additional variables:

\[ \omega = (y_3 - y_1)(x_4 - x_2) - (y_4 - y_2)(x_3 - x_1) \]

\[ \xi = -(y_3 - y_1)(x_4 - x_2) - (y_4 - y_2)(x_3 - x_1) \]

\[ \Delta = (x_3 - x_1)(y_4 - y_2) - (y_3 - y_1)(x_4 - x_2) \]

The parameters are then given by

\[ \lambda = \frac{\omega}{\Delta} \quad \mu = \frac{\xi}{\Delta} \]

If 0 < \lambda < 1 and 0 < \mu < 1, the coordinates of the intersection are given by the equation:

\[ (x, y) = \frac{(x_1, y_1)\lambda + (x_2, y_2)\mu}{\lambda + \mu} \]
7. Spatial Operation

This method is very efficient because it requires only eight multiplications and one division to determine whether two line segments intersect.

Multiplication and division require more computing time than addition and subtraction. Evaluation of computation efficiency is thus based on the number of multiplications and divisions.

7. Spatial Operation

Calculation of polygon area

Spatial overlay calculates the area of polygons and adds as an attribute of new polygons.

The area of polygon represented by a chain of coordinates \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\) is given by

\[
S = \frac{1}{2} \sum_{i=1}^{n} (x_i, y_i) - (x_{i+1}, y_{i+1})
\]

7.3 Spatial search

Spatial search is a kind of search operation used in database management.

Spatial search differs from other search operations in that it uses the location of spatial objects as a search key as well as attribute data.

Search method depends on the type of spatial objects.

7.3.1 Search from points

1. Find the nearest spatial object (point, line, or polygon) from a point, or a set of points.
2. Enumerate all the spatial objects located within a certain distance from a set of points.

3. Find the polygons that contain a set of points in their inside.

7.3.2 Search from lines

1. Find the nearest spatial object from a line, or a set of lines.

2. Enumerate all the spatial objects located within a certain distance from a set of lines.

3. Find the polygons that contain a set of lines in their inside.

4. Enumerate all the spatial objects (lines or polygons) that intersect a set of lines.
7. Spatial Operation

7.3.3 Search from polygons

1. Find the nearest spatial object (point, line or polygon) from a polygon, or a set of polygons.
2. Enumerate all the spatial objects (points, lines or polygons) located within a certain distance from a set of polygons.
3. Find the polygons that contain a set of polygons in their inside.
4. Enumerate all the spatial objects (lines or polygons) that intersect a set of polygons.

7.3.4 Basic algorithms used in spatial search

Search operations use various computational algorithms to answer geometrical questions such as
1. point-in-polygon problem, and
2. point-location problem.

Point-in-polygon problem

Point-in-polygon problem asks, given a point and a polygon, whether the point is contained in the polygon. This is a basic question in spatial search.

Plumb-line algorithm

Plumb-line algorithm is a simple but efficient algorithm for point-in-polygon problem.

Given a point and a polygon, plumb-line algorithm draws a vertical half line from the point downward, and count the number of intersections of the line and the polygon.

If the number of intersections is odd, the point is contained in the polygon. Otherwise, the point is not contained in the polygon.
This algorithm works efficiently for disconnected polygons and polygons with holes.

7. Spatial Operation

Point-location problem

Point-location problem asks to find the polygon that includes a given point. This problem occurs, for example, when we want to know the address of a point.

A simple algorithm is to examine individual polygons one by one using plumb-line algorithm whether it contains the point. This algorithm, however, is not efficient, because it may have to examine all the polygons to find the answer.

Slab method is a hierarchical search algorithm, which runs faster than the simple method.

As a preprocess, slab method divides the plane by horizontal lines passing the vertices of polygons into slabs, and records the intersecting polygons in each slab.

Slab method searches the slab that contains the point, and then the polygon in the slab that contains the point.

Search of the slab that contains a point is a one-dimensional search along the vertical axis (for one-dimensional search, see Sedgewick (1988)), so it is even simpler than plumb-line algorithm. Since plumb-line algorithm is applied only for a limited number of polygons, slab method runs faster than the simple method.

Slab method requires considerable space for storing the slab data. In spite of this disadvantage, slab method is widely used in GIS because of its computational efficiency.
Bucket method

Bucket method is an extension of slab method to two-dimensional space.

Bucket method divides the plane by a rectangular lattice as a preprocess.

Bucket method then searches the cell that contains the point, and apply plumb-line method in the cell to find the polygon that contains the point.

As well as slab method, bucket method is a hierarchical search algorithm.

In slab method, slab size depends on the number of vertices of polygons. In bucket method, on the other hand, bucket size is not limited by polygon number. Consequently, if we choose an appropriate bucket size, bucket method runs as fast as the slab method, and requires less space for data storage.

7.4 Buffer operation

Buffer operation generates a set of polygons that represent the regions within a certain distance called ‘buffer distance’ from a set of spatial objects.

Buffer operation can be applied to points, lines and polygons.

Buffer distance can differ among spatial objects.

Figure: Buffer operation for points

Figure: Buffer operation for lines
I analyzed with an undergraduate student the spatial distribution of crimes in Funabashi, Chiba in 1999. We collected crime data from January, 1999 to December 1999, paper maps showing the location of crime occurrences. We applied buffer operations to spatial data of urban facilities and performed statistical analysis to detect significant patterns of crime occurrence.

We found the following crime patterns:

1. Snatches are clustered on roads of 5-13 meters wide.
2. Burglars prefer to the houses located within 15 meters from parking lots.
3. Thieves tend to choose cars parking on the riverside.

Voronoi diagram is a spatial tessellation generated from the distribution of spatial objects. Assigning every point to its nearest spatial objects, we obtain a Voronoi diagram.

Voronoi diagram for points consists of bisectors of lines connecting points.
7. Spatial Operation

7.5.1 Construction of Voronoi diagram

The simplest method of constructing Voronoi diagram is to follow directly its definition as a computational algorithm, that is, to calculate the intersections of half planes given by the bisectors of lines connecting points. However, this method is obviously inefficient, so faster algorithms are used.

http://www.voronoi.com/ImplFrames.htm

Incremental method

Incremental method is most widely used for constructing Voronoi diagram in GIS. Incremental method first constructs Voronoi diagram for two points. Then it adds another point and modifies the diagram to obtain Voronoi diagram for three points. Adding points one by one, incremental method gives Voronoi diagram for a given set of points.

7.5.2 Applications of Voronoi diagram

Voronoi diagram is used

1. to divide a region into subregions by assigning every location to its nearest spatial object,
2. to approximate the catchment area of urban facilities such as post offices and convenience stores, and
3. to find the most inconvenient location, the location whose distance to the nearest spatial object is largest.

The third problem is called ‘the largest empty circle problem’.

• References - Voronoi diagram

In addition to those direct use, Voronoi diagram also works as a computational tool for other spatial operations.

If we have to search frequently the nearest spatial object, it is efficient to construct Voronoi diagram and store the data explicitly.

We can find the nearest object by applying the point-in-polygon algorithm to Voronoi diagram. This is faster than to calculate the distance between spatial objects every time the search is requested.

Moreover, Voronoi diagram is also used in spatial interpolation. Discrete interpolation uses Voronoi diagram because it substitutes the function value at a certain location by that of its nearest sample point.

Voronoi diagram is defined for not only points but also lines and polygons.

Computational algorithms are more complicated than those for usual Voronoi diagrams, so commercial GIS does not have a function of constructing Voronoi diagrams for lines and polygons.

Delaunay triangulation, which is used in spatial interpolation, consists of line segments that connect point pairs whose Voronoi regions are adjacent.

This implies that we can construct Delaunay triangulation from a Voronoi diagram, and vice versa. We thus say Delaunay triangulation is the dual graph of Voronoi diagram.
7. Spatial Operation

7.6.1 Construction of Delaunay diagram

We can construct Delaunay triangulation of points from their Voronoi diagram, following directly its definition.

Another method is to use another property of Delaunay triangulation, that is, any circumcircle of Delaunay triangles does not contain other points in its inside.

1. Draw the circumcircle of a set of any three points.
2. If the circle does not contain other points in its inside, connect the three points by a triangle.
3. Repeat the above two steps for all the sets of three points.

Unfortunately, this algorithm does not run so fast. Therefore, in GIS, more efficient algorithms are used.

7.6.2 Properties of Delaunay triangulation

As shown above, Delaunay triangulation has several properties that distinguish it from other triangular irregular networks (TINs).

1. Voronoi polygons whose generators are connected by an edge of a Delaunay triangle are adjacent.
2. Any circumcircle of Delaunay triangles does not contain other points in its inside.
3. Delaunay triangulation consists of triangles close to equilateral triangles.

Suppose all the possible triangular irregular networks defined for a set of points. We sort the minimum angles of individual triangles in ascending order. Among all the triangulations, Delaunay triangulation gives the maximum angles for each rank. This implies that the minimum angles of Delaunay triangles are larger than those of other triangles. Delaunay triangulation does not have elongated, acute-angled triangles.

This property is desirable for dividing a region into subregions in spatial interpolation. Delaunay triangulation is thus used for defining subregions in spatial interpolation, given a set of sample points.

7.6.3 Applications of Delaunay triangulation

Delaunay triangulation is used
1. to divide a region into subregions in spatial interpolation and the finite-element method,
2. to give a natural definition of ‘neighboring points’ in image processing and perceptual psychology, and
3. to construct the minimum spanning tree, a set of connected line segments of the minimum length that connect all the given points (the minimum spanning tree is a subset of Delaunay triangulation).

7.7 Network operation

Line data representing traffic roads, railways, and pipelines, store topological information explicitly, that is, we can tell whether two lines (arcs) are directly connected.

This type of line data are often called ‘network’ data. Network data are distinguished from ordinary line data, though topological information can be generated from ordinary line data in GIS.
In network data, points are usually called ‘nodes’, and lines are called ‘links’ or ‘edges’.

7. Spatial Operation

Network analysis is a set of spatial operations performed on a network data. It includes
1. shortest path tree problem,
2. maximum flow problem,
3. spatial search and buffer operation on a network,
and so forth.

7.7.1 Shortest path tree problem

Shortest path tree problem is to find the shortest path between two nodes on a network.

Car navigation systems solve this problem and show us the best (shortest) route to our destination.

Dijkstra’s algorithm (label-setting method)

To solve shortest path tree problem, most systems including GIS and car navigation systems use a very famous computational algorithm called “Dijkstra’s shortest path algorithm” developed by E. W. Dijkstra in 1959. Though it has been gradually improved in literature, Dijkstra’s basic idea is still widely used in GIS and operations research.
Dijkstra’s algorithm searches for the shortest path from the starting node to every other node one by one. It halts when the shortest path from the starting node to the destination node is found.

Dijkstra’s algorithm first searches for the shortest path from the starting node to nodes directly connected to the starting node. The search gradually expands from the starting node to outer nodes.

Our objective is to find the shortest path from the starting node 1 to the destination node 5.

Every node has three variables.

1) $p_i$: Length of the temporary shortest path from the starting node to node $i$

This variable is initially set to $\infty$ for all the nodes except the starting node, because the shortest path is not known before the search starts. It is set to 0 for the starting node.

The variable is updated with the progress of the search.

2) $e_i$: ID number of the node from which the temporary shortest path to node $i$ comes

This variable shows the information about the temporary shortest path to node $i$ comes.

Temporal shortest path

3) $s_i$: State of the search process of the shortest path to node $i$

Information about the shortest path to node $i$ is updated until the search finishes. While the search still continues, that is, it is still possible to find shorter paths to node $i$, the variable $s_i$ is set to 0. If the temporary path is proved to be the shortest one, the variable is set to 1. Initially, it is set to 0 for all the nodes including the starting node.

We can retrieve the temporary shortest path from the starting node to node $i$ by tracing the variables $e_i$ back to the starting node.

This variable is initially set to $\emptyset$ for all the nodes except the starting node, and * (asterisk) for the starting node.

It is updated with the progress of shortest path search.
The three variables are represented as a 3-tuple.
\((p_i, e_i, s_i)\)

The search halts when \(s_i = 1\) for the destination node.

### Initial setting

Variables for node \(i\): \((p_i, e_i, s_i)\)

1. \((0, *, 0)\)
2. \((\infty, \emptyset, 0)\)
3. \((\infty, \emptyset, 0)\)
4. \((\infty, \emptyset, 0)\)
5. \((\infty, \emptyset, 0)\)

### Outline of Dijkstra’s algorithm

1. If \(s_i = 1\) for the destination node, the search halts.
2. Otherwise,
   i. Find the node of the smallest \(p_i\) among the nodes whose \(s_i\) is 0. We call it the node under consideration.
   ii. Examine every node directly connected to the node under consideration, and update its \(p_i\) if it is larger than the length of the path by way of the node under consideration.
   iii. Set \(s_i\) of the node under consideration to 1.
   iv. Go to step 1.

### Step 2-i

Dijkstra’s algorithm finds the node of the smallest \(p_i\) among all the nodes of \(s_i = 0\).

1. \((0, *, 1)\)

The shortest path for the node found is fixed at this time.

### Step 2-ii

Dijkstra’s algorithm examines every node directly connected to the node under consideration, and updates its \(p_i\) if it is larger than the length of the path by way of the node under consideration.

Dijkstra’s algorithm first examines node 2. The length of its temporary shortest path \((p_2)\) is 8, while that of the path by way of node 3 is 6+5=11. Since the path by way of node 3 is longer than the temporary shortest path, \(p_2\) is not updated.

Dijkstra’s algorithm then examines node 4. The length of its temporary shortest path \((p_4)\) is \(\infty\), while that of the path by way of node 3 is 6+3=9. Thus \(p_4\) and \(e_4\) become 9 and 3, respectively.
The variables then change from

1 \( (0, *, 1) \) to

1 \( (0, *, 1) \)

Our objective is to find the shortest path from the starting node 1 to the destination node 5.

We can retrieve the shortest path from node 1 to node 5 by tracing the variable \( e_i \) back from node 5 to node 1.
Properties of Dijkstra’s algorithm

1. Dijkstra’s algorithm detects the shortest path for two given nodes, and also shows the shortest paths from the starting node to other nodes as by-products.
2. Search area gradually expands from the starting node to all directions. Even if we want to know the shortest path from Hongo campus to Komaba campus, Dijkstra algorithm searches in the east and north regions of Hongo campus. This property is undesirable in the practical use of the algorithm, because it searches for unnecessary shortest paths.

A* algorithm

To overcome the second (undesirable) property of Dijkstra’s algorithm, a new algorithm called ‘A* algorithm’ was developed. It is an extension of the original Dijkstra’s algorithm.

It often happens that the lower bound for the length of the shortest path can be calculated. When we define node length by the network distance, the lower bound of a path is given by the Euclidean distance between the starting and destination nodes.

A* algorithm uses the information about the lower bound of the shortest path length in order to limit the search area.

Let $b_i$ be the lower bound for the length of the shortest path from node $i$ to the destination node. It is usually given by the Euclidean distance between the two nodes.

Instead of $p_i$, A* algorithm uses $p_i + b_i$ at Step 2-i to choose the node under consideration. This is because the shortest path is unlikely to pass the nodes of large $p_i + b_i$ values. A* algorithm searches the nodes of small $p_i + b_i$ values earlier than those of large values.

A* algorithm runs even faster than the original Dijkstra’s algorithm.

Note, however, whether A* algorithm works successfully depends on the calculation of $b_i$. If it is not given appropriately, A* algorithm is less efficient than the original Dijkstra’s algorithm. A* algorithm fails when overpasses and underpasses exist in network data.

A algorithm

When the lower bound cannot be calculated, it is often replaced by its approximation. A algorithm is the same as A* algorithm except that it uses an approximation of the lower bound in shortest path search.

A algorithm runs faster than A* algorithm, but it is not guaranteed that the result is optimal.
7.7.2 Maximum flow problem

Maximum flow problem is to calculate the maximum flow on a network where the capacity of links is given.

The maximum flow problem appears in
1. planning of water supply network,
2. analysis of traffic congestion, and
3. planning of evacuation route.

Since the explanation of network flow theory and its solutions requires considerable time, I skip the details of network flow problem. Instead, I recommend two books about network flow analysis, one of which was written by a famous Japanese researcher in operations research.

7.7.3 Spatial operations on a network

Some spatial operations can be performed on network as well as on a plane.

References - network flow
7. Spatial Operation

Spatial search on a network - an example

Find the nearest point on a network from a given point.

Buffer operation on a network - an example

show the area within a certain distance from a given point.

Spatial operations on a network is important in spatial analysis at a local scale such as analysis of the human behavior in urban areas, because spatial phenomena are observed at local scale on traffic networks.

The term ‘micromarketing’ refers to marketing strategy which focuses on the microscale human behavior in urban areas. In micromarketing, spatial data are analyzed on a network. In GIS we often say that we have to move ‘from global to local’ analysis.

Figure: Market analysis on a network

Homework Q.7.1 (10 pts)

Show several methods of calculating the gravity center of a polygon whose vertices are represented by a chain of coordinates \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\), and compare the efficiency of their computation.
Homework Q.7.2 (10 pts)

1) Place five points on a plane and construct the Voronoi diagram for the points.

2) Describe the Voronoi diagram by the arc-node data structure (indicate the coordinates of node \(i\) as \((x_i, y_i)\)).

3) Add two points to the diagram by the incremental method.

Homework Q.7.3 (10 pts)

Choose two nodes in the network below and find the shortest path between the nodes by Dijkstra’s algorithm.