A Parallel Line Segment Intersection Strategy Based on Uniform Grids

ZHOU Qin  ZHONG Ershun  HUANG Yaohuan

Abstract  The line segment intersection problem is one of the basic problems in computational geometry and has been widely used in spatial analysis in Geographic Information Systems (GIS). Lots of traditional algorithms study the problem in a serial environment. However, in GIS, a spatial object is much more complicated and is considered to be always composed of multiple line segments, and one line segment connects another line segment at its endpoint. On the other hand, along with the advances made in computer hardware, more and more personal computers have multiple cores or CPUs equipped. Thus, to make full use of the increasing computing resources, parallel technique is applied as one of the most available methods. Apparently, the traditional algorithms should be improved to take advantage of the technologies. Under these circumstances, based on the modified uniform grid algorithm, which is adapted to dealing with spatial objects in GIS, this paper proposes a parallel strategy in a shared memory architecture. Also, experimental results are given in the final part of this paper to demonstrate the efficiency this strategy brings.

Keywords  line-segment intersection; parallel computing; GIS

CLC number  P208

Introduction

Reporting of segments set intersections is one of the fundamental problems of computational geometry. Line segment intersection detection and reporting algorithm is useful in GIS, such as map overlay, spatial join in spatial databases, Boolean operations, and so on. It is an operation of great computational complexity and has been extensively studied to accelerate the computing process.

However, sometimes, it cannot satisfy the GIS applications, especially when dealing with the ever-increasing size of dataset. With the development of computer hardware, personal computers equipped with multiple cores or multiple CPUs become more and more popular. Furthermore, GIS applications are often distributed on servers with high CPU capabilities. Such computers provide a strong basis for faster parallel data processing. In many fields, researchers have applied parallel computing technology to practical problems and made considerable achievements. On the other hand, in GIS, especially in GIS platform software, traditional algorithms in a serial environment are still the main stream and cannot make full use of these compute resources. Therefore, research on parallel strategy of these problems is indispensable.

Given this situation, we are motivated to study on the parallel strategy to make full use of the compute resources to accelerate the computing process. Based on uniform grid algorithm [1], we focus on the
shared memory parallel environment, in order to make applications perform well on the most popular computers that are equipped with multiple cores or multiple CPUs. The paper proposes a regular grid-based line intersection algorithm, which is parallelized after the domain decomposition of the study area.

1 Background and related work

The line segment intersection problem is defined as follows: given a set $S$ consisting of $n$ segments in the plane, find all intersecting pairs and all the $k$ intersection points.

With a simplified insight, an object-oriented method can be applied. There can be up to $O(n^2)$ intersection points, since if every segment is intersected by each of the other segments, there would be $(n-1) + (n-2) + \cdots + 1 = n(n-1)/2 = O(n^2)$ intersection points. In the worst case, to compute them all would require an $O(n^2)$ algorithm. The “brute force” algorithm would simply consider all $O(n^2)$ pairs of line segments, test each pair for intersection, and record the pairs found. This is a lot of computing. However, in the case that there are only a few intersection points or that only one such point needs to be detected (or not), there should be faster algorithms.

In fact, these problems can be solved by “output-sensitive” algorithms whose efficiency depends on both the input and the output sizes. Here, the input is a set of $n$ segments, and the output is the set of $k$ computed intersections. Since the 1970s, the researchers in GIS, computer graphics, and computational geometry have made great progress with regards to fast line segment intersections. These researchers have developed several effective intersection algorithms, which basically avoid unnecessary chain-pair checking. The typical algorithms include the plane-sweep algorithms, band algorithm, uniform grid algorithm, and Delaunay triangulation.

An early algorithm$^{[2, 3]}$ showed how to detect if at least one intersection exists in $O(n \log n)$ time and $O(n + k)$ space$[^4]$. Later, Chazelle and Edelsbrunner discovered an optimal $O(n \log n + k)$ time and $O(n + k)$ space algorithm$[^5]$. Mulmuley, using randomized approach, offered an algorithm with expected running time $O(n \log n + k)$ and space requirement $O(n + k)$$[^6]$. Clarkson and Shor’s algorithm also based on the randomized approach has expected running time $O(n \log n + k)$ and space requirement $O(n)$$[^7]$. The traditional algorithms mentioned above are implemented in a serial environment. They concentrate on optimizing the running time and space requirement, which may come to an extreme. With the development of the computer hardware, the personal computers and the application servers are equipped with more and more computing resources. Since sequential algorithms is not capable of making full use of the increased resources provided by multicore or multi-CPU computers, the parallel method should be applied to improve this situation.

Up to now, there are some related parallel algorithms. The uniform grid algorithm$[^1]$ expands the sweep line from one dimension to regular rectangles and makes the algorithm parallelizable on parallel machine. However, it is not robust enough to deal with the irregular distributed data. The band algorithm based parallel algorithm$[^2]$ is applied in vector overlay.

Since we aim at improving the use-ratio of the computing resources, the parallel algorithm is implemented by multithreading, which is based on the shared memory parallel computing model. Some problems associated with the distributed parallel systems will not be taken into account, such as the communication cost between subsystems. What we mainly stress is the load-balance among the threads implementing computing tasks. Moreover, by domain decomposition, the data skew attained by irregular data distribution is relieved.

2 Parallel strategy

The parallel processing is a class of computerized information processing that emphasizes the concurrent manipulation of data segments or concurrent execution of process components to solve a problem or to accomplish a task, often on a specially designed
computer. Using Flynn’s classification, the common parallel computer architectures fall into two categories: SIMD (single instruction stream, multiple data stream) and MIMD (multiple instruction streams, multiple data streams)\[8\]. The parallel technique is adopted to achieve higher availability and better performance, in any case. The system can execute subtasks concurrently. The speedup that we can achieve with parallel programming depends on the number of processors available and the extent to which workloads are balanced among the processors and the overhead for parallel computing. Therefore, we must choose the right parallel programming technique so that the system can distribute the subtasks as evenly as possible among the processors and make the application more efficient. The data partition and function partition are effective parallel programming techniques for most applications.

In the parallel line segment intersection algorithm research, we simulate SIMD architectures, say, SPMD (single program, multiple data stream) architectures, because the spatial model of the line segment intersection is a regional scope problem \[9\]. Therefore, the data partition by domain decomposition strategy is one of the most important factors affecting final running time when dealing with the ever-increasing dataset size in GIS, which will be discussed in Section 2.2. The outline of line segment intersection parallel strategy is the DCSO (Decompose computing task, Conquer subtasks, Stitch subresult, and Output final result) strategy. DCSO is used in GIS parallel strategy in common.

Before addressing parallel processing, Section 2.1 describes the uniform grid-based line segment intersection algorithm, including the data structure and the algorithm steps, which can be considered as the instruction collection for each subtask to be down in sequence. Section 2.2 gives the parallel strategy. First, the work flow of the parallel process is introduced. Then, some related algorithms are described, including the spatial data domain decomposition algorithm and the stitch approach. The load-balance problem is emphasized in the spatial domain decomposition step. Section 2.2.3 is the stitch method dealing with the patches resulting from the previous step.

### 2.1 Sequential algorithm

The uniform grid based on the algorithm extends the classical line sweep object from a vertical line to a 2-D rectangle in the plane. The sweep step size is the size of the rectangle. Therefore, at the beginning of the algorithm, the study area is covered by sweep rectangles of regular size. Assume that \( N \) line segments are distributed in \((S_x \times S_y)\) study area. We place \( M \times N \) grids to cover the area. The uniform grid size is \( \frac{S_x}{M}, \frac{S_y}{N} \). The grid cells partition the study area without overlaps or omissions. The intersection algorithm is described as follows.

The classical algorithms are dealing with line segments. However, in GIS, the spatial objects are much more complicated. For example, a spatial object may be composed of several simple objects with identical attributes, and each subobject is a chain composed of many points; the object maybe intersecting with itself; and so on. To make the uniform grid algorithm available, we should define more complex data structures.

#### 2.1.1 Data structures

In the uniform grid algorithm, the grid is the basic unit that makes the algorithm parallelizable by Franklin in parallel machine. When dealing with spatial objects in GIS, we record more information in the grid cell. Thus, the structure of the grid unit is designed as \( G\{cell, segments\} \), in which we have the following:

1) \( cell \) is the grid identifier defined as \( l=G_x \times G_y \). \( G_x \) is the grid number in the \( x \)-direction and \( G_y \) in the \( y \)-direction.

2) \( segments \) is the collection of the subparts of the line object that intersect with the grid with identifier \( cell \).

The two collections are composed of the following: 

- \( LineSegments \) is an array filled with the coordinates of the points of all the line objects in the study area.
- \( IntersectPoints \) is the collection of all the intersection points.

A single segment is designed as \( Segment\{object-id, next-vertex-no, start-point-no\} \), in which

1) \( object-id \) is the identifier of the line object;
2) next-vertex-no is the index number for the end point of the subline intersecting with the special grid;
3) start-point-no is the index number for start point of the subline intersecting with the special grid in the LineSegments array; through start-point-no, the point coordinates on the subline can be found easily.

For a Segment, it does not necessarily satisfy start-point-no=next-vertex-no − 1, since the next-vertex-no is the point’s index on the line object and start-point-no is the point’s index in a collection of points of all the line objects, say LineSegments. However, for the first line object, the equation must be satisfied.

For each line object, the intersection point is recorded as LineSegmentIntersects (object-id, intersect-indexes, point-indexes), in which
1) object-id is the identifier of the line object;
2) intersect-indexes is the collection of intersection point index in IntersectPoints;
3) point-indexes is the collection of the end point index of the line segment on the line object associated with the next-vertex-no in Segment.

Therefore, intersect-indexes is the collection of all the intersection points’ index numbers on the special line object with identifier object-id. The intersection points can be gotten through the index in intersect-indexes in IntersectPoints.

### 2.1.2 Algorithm steps

1) The 2-D sweep process. The output of this process is a collection of G. For each segment in LineSegments, it determines which grid it passes through, and the grid is recorded in the form $G(cell, segments)$. In the best situation, each grid associates with the same number of segments. Therefore, we give an empirical value for the grid size: $M \times N = \frac{G_s}{C \times 0.8} \times \frac{G_s}{C \times 0.8}$, in which $C$ is the count of the objects in the study area.

2) Intersection computing process. For each grid $G(cell, segments)$, the intersections are computed and recorded in IntersectPoints and LineSegmentIntersects. In addition, if $size(segments) < 2$, there is definitely no intersection in the grid $G$.

3) Rearrangement of the line objects. For each line object, the intersection points can be indexed from LineSegmentIntersects and IntersectPoints. Therefore, the line object is rearranged by inserting the intersection points into the certain line, and the line object is clipped without extracting all the segments.

The workflow of the algorithm is shown in Fig.1. The italic text is the output of the step, and the meaning is explained in Section 2.1.1.

![Fig.1 Workflow of the grid-based line-intersection computing](image)

### 2.2 Parallel strategy

#### 2.2.1 Workflow of the parallel strategy

For the standard uniform grid based on algorithm dealing with simple line segments, it is easy to make it parallelized in the grid sweeping process and the intersection computing process. But the sequential algorithm proposed in Section 2.1 deals with the complex spatial objects in GIS, and the processes for each object and each grid are associated with global structures such as LineSegments and IntersectPoints to ensure the object clipped unless there are intersection points on it and the line segments composing the original object are still the sub-parts of the object unless there are intersection points on the segment. For this kind of problem, the domain decomposition is a beautiful strategy. The work flow is shown in Fig.2.

![Fig.2 Workflow of the parallel intersection computing](image)
The parallel strategy adopts the DCSO method:
1) Decompose computing task. Decompose the study area into several subparts. The count of objects in each part is approximately the same. The bounds of the subparts may be overlapped.
2) Conquer subtasks. Apply the serial algorithm described in Section 2.1 for each subpart. In the overlapped area, the objects from different subparts may have intersection points. Therefore, the result of each part falls into two kinds: the final result and the stitch data.
3) Stitch subresult. For the stitch data resulting from Step 2, the objects with different subpart ID are intersected again using the serial line-intersect algorithm.
4) Output final result. The combination of the final result in Step 2 and the result in Step 3 is the final result.

2.2.2 Domain decomposition algorithm

The spatial objects are different from common data in that they are multidimensional and are co-relational in the space, which means that the longer the distance between two objects, the fainter the influence is. The study area is decomposed into an appropriate number of subdomains. Each subdomain is a self-governing area. The objects that step into more than one subdomain need special treatment for they may intersect with the objects in the neighborhood subdomains.

The domain decomposition is also one of the most effective solutions for the load-balance problem. Making the subdomains contain approximately the same number of objects is rather important. In our strategy, the objects are not clipped, so a single object only belongs to one subdomain, and the subdomains may intersect with others.

However, sometimes, the successful decomposition according to the objects count in each subdomain will not achieve the perfect load-balance for a parallel algorithm because the algorithm is in \(O(f(n))\) time, in which \(n\) is the number of points but not the number of objects. If the point density on different objects varies greatly, which means the count of the points composing the line or region object differs greatly, the object oriented domain decomposition strategy is meaningless for an \(O(f(n))\) algorithm. Therefore, we assume that the count of points on different object is nearly the same.

As mentioned above, the spatial data will be partitioned by range to preserve the continuity in space. Therefore, it is essential to split the boundary \((P)\) into several parts \((P_1, P_2, \ldots, P_n)\).

Supposing that each part \(P_i\) contains \(C_i\) objects and \(C\) is the total number of the objects in the dataset, the objects in \(n\) subparts compose one division of \(C\). That is, \(C\) and \(C_i (1 \leq i \leq n)\) must satisfy two conditions: \(C = C_1 \cup C_2 \cup \ldots \cup C_n\) and \(C \cap C_1 \cap \ldots \cap C_n = \emptyset\).

Different from \(C_i\), the boundary of the subspace \(P_i\) is not necessarily a division of the space \(P\), which means that the union of \(P_i\) may or may not cover the whole space of the dataset, and the intersection of \(P_i\) may or may not be empty. However, \(P_i\) is the union of the MBRs of the objects in \(C_i\).

In order to achieve load-balance in the process of parallel line objects intersection, the number of objects in each part should be the same theoretically as we deploy the research in an isomorphic environment, and \(C_1 = C_2 = \ldots = C_n\) is what we want. However, the spatial objects cannot be previewed, and the status of the dataset cannot be right in hand, such as the data distribution in the space, especially when the dataset size is massive. Therefore, an algorithm is needed to split the dataset into \(n\) parts beforehand, which will be explained in Fig.4.

Suppose \(n=4\), Fig. 3 shows how the spatial data is partitioned and each part will be processed by a single thread. The objects intersecting with the split lines can be resided in a part according to the area split by

![Fig.3 Spatial dataset decomposition in four parts](image-url)
the split line. The spatial objects within the dataset boundary will be divided into four parts \( (P_0, P_1, P_2, P_3) \), and each part is tuneable to contain approximately the same number of objects.

Here, we adopt a sweep line algorithm to decompose the study area. The scan line structure is designed as group \( \text{Scanline}(index, C_{RB}, B_{RB}) \), in which the index is the position of the scan line, \( C_{RB} \) is the object count in the right-or-bottom direction of the scan line, and \( B_{RB} \) is the union of the bounds of the objects in left-or-top direction.

The dataset is parted into \( 2^{2n} \) parts by \( \sum_{i=0}^{n} 3 \times 4^{i-1} \) scan lines. For the selection of single scan line in a loop, first, a vertical split line \( S_v \) is selected; then, two horizontal split lines \( S_{HR} \) and \( S_{HL} \) are selected for the left and right part of \( S_v \). The algorithm is described as follows:

1. Initialization
   - Scan line count: \( C_{scan} = \sqrt{(C)} \)
   - Average count: \( C_{average} = C_{scan}/2^n \)
   - Vertical scan line step: \( S_{v} = G_v/C_{scan} \)
   - Horizontal scan line step: \( S_{h} = G_h/C_{scan} \)
   - Bound: \( \text{Bounds of the study area} \)

2. Select vertical split line \( S_v \) as follows
   - Create scan lines: \( \text{scanlines} = \text{Scanline}(C_{scan}) \)
   - For each \( \text{Scanline} [j] \) \( (0 < i < C_{scan}) \), \( C_m = B_m = 0; \)
   - For \( (i=0;i<C_{scan};i++) \)
     - \( / \) get the left and right scan line covered by the MBR of the object/*
       - \( / \)\( S_{vleft} = (\text{ltmBound.left-Bound.left})/S_{v} \)
       - \( / \)\( S_{vright} = (\text{ltmBound.right-Bound.right})/S_{v} \)
     - \( / \) Scan line in center/*
     - \( / \)\( S_{center} = (\text{ltmBound.left} - 2*\text{Bound.left})/(2*S_{v}) \)
     - \( / \)\( S_{left} = (\text{ltmBound.right} - 2*\text{Bound.right})/(2*S_{v}) \)
     - \( / \) Scanline[j], \( C_m++; \)
     - \( / \) Scanline[j], \( B_m = \text{Union(MBR)}; \)
     - \( / \) Scanline[j], \( C_m = C_{average}; \)
     - \( / \)\( S_{v} = \text{Min(abs(scanline[j], C_m-C_{scan}));} \)
   - 3. Select horizontal split lines \( S_{HL} \) and \( S_{HR} \) simulate to step2.

Fig.4 Scan-line-based decomposition algorithm

When \( 2n \) scan lines are selected by \( n \) loops, the subdomains are determined. As defined above, it is easy to find that the bounds of the subdomains satisfied: \( P_i = \text{Scanline}[i] \times B_{RB} \) \( (0 \leq i \leq n - 1) \).

2.2.3 Patches stitch

We define patch as the intersection area among different subdomains. In the patches, the objects from different subdomains are not processed in the parallel computing step. Therefore, after parallel computing for subdomains, the patches stitch is the sequential work.

In the progress of the subdomains processed, respectively, the result objects intersecting with certain patches are recorded as a collection of \( \text{Patches} \) \( (id) \), which is the collection of the objects need to produce intersections that has been lost in the parallel progress. These objects will be applied with the sequential algorithm when two objects are from different subparts.

3 Experimental results

3.1 Choose platform of parallel computing

Originally, the parallel computing runs on MPP (Massively Parallel Processors), SMP (Symmetric Multiprocessors), or clusters. With the rapid development of commercial personal computers (PCs), the parallel programs can run on PCs equipped with multiple cores to make full use of the computing resources. The same with the method on the SMP computers, the parallel programs on multicore PCs also adopt multithreading. OpenMP is widely used in multithreading applications, and many commercial compilers support it.

In the experiment, the parallel program is written in \( C++ \), and the parallel subwork is accomplished by multithreading based on OpenMP.

3.2 Experimental environment

The experimental data is part of the Chinese hydrology data mixed with some grid tile lines. The count of the spatial line objects is 178 876 with 2 925 832 points. The distribution of the spatial objects is irregular. Generally speaking, there are much more lines, which represent the rivers or lakes. For the irregular distribution spatial dataset, the spatial domain decomposition algorithm is applied, and the effectiveness of our strategy is tested.

The computer used to do the experiment is a common dual-core personal computer, Inter(R) Core (TM) 2 CPU, 1.8GHz, and 1.99GB memory. The operating system is Microsoft Windows 2003 Enterprise Server Edition, (05.02.3790.00). The spatial data is stored in Oracle (Oracle 10g) through spatial database engine.
3.3 Analysis of the results

3.3.1 Domain decomposition results

By using the algorithm described in Section 2.2.2, the spatial dataset is decomposed into four subdomains. The objects count in each subdomain is listed and presented in Table 2, and the status of the decomposed dataset is shown in Fig. 5. \( S_V \) is the vertical split line, and \( S_H L \) and \( S_H R \) are the horizontal split lines according to the algorithm in Section 2.2.2. The split lines make the subdomains contain approximately the same count of spatial objects, which is very important to the load-balance in parallel computing.

![Spatial domain decomposition](image)

3.3.2 Time-consuming analysis

In parallel computing process, there are three main steps, which consist of the processing time in a single subdomain according to the algorithm described in Section 2.1.2, see Eq. (1).

\[
T = T_G + T_I + T_R
\]  

(1)

Where \( T_G \) is the time spent on the grid sweeping step, which is described in Step 1 in Section 2.1.2. \( T_I \) is the time spent on line segments intersection in all grids. \( T_R \) is the time spent on points rearrangement.

In a single thread, the subwork is completed in \( O(n+k) \) time, in which \( n \) is the number of points, and \( k \) is the number of intersection points. \( T_G \) is affected by \( n \), \( T_R \) is affected by \( (n+k) \), and \( T_I \) is affected by \( k \).

In our experiment environment, \( T_G \) and \( T_R \) are determined by system I/O. Several threads accessing spatial data concurrently will not obtain higher performance. \( T_I \) is the most CPU-consuming operation. Therefore, more \( T_I \) time will make the parallel computing more effective, which means comparing with the number of points \( n \), the more the intersection points are, the better the parallel speed-up ratio is.

3.3.3 Parallel computing results

In the sequential computing process, only one thread is activated to perform the sequential algorithm, and the time consumed is listed in Table 1. In the parallel computing process, two threads started to process different subdomains of the dataset, and the results are presented in Table 2.

| Sub Part | Objects | Thread-number | Points | CPU time(s) | \( C_I \) |
|---------|---------|---------------|--------|-------------|---------|
| 0       | 44,967  | 0             | 799,776| 68.628      | 369,919 |
| 1       | 44,914  | 1             | 782,192| 65.042      | 469,135 |
| 2       | 44,454  | 0             | 681,248| 66.149      | 325,171 |
| 3       | 44,541  | 1             | 662,616| 61.550      | 387,776 |

Note: \( S_T \) is the stitch time; \( T \) is the total time; \( C_I \) is the count of the intersection points.

The speedup rate of the parallel computing here is 1.39. In theory, in two threads parallel environment, the speedup rate would be equal to 2.0. The rate leans from the theoretical value, because the most CPU consuming operation in the algorithm is only part of the subcomputing and the patches stitch operation, which is necessary, and also occupies pieces of time.

As a whole, the strategy performs better than the sequential one. Moreover, the spatial dataset domain decomposition algorithm ensures the load-balance between threads.

4 Conclusion

GIS applications usually deal with large amounts of spatial data, and some basic spatial operations are time consuming. The parallel computing is one of the most effective methods to solve this problem. In ad-
dition, there comes the mini parallel computing environment—the multicore personal computers.

In this article, the intersection operation on spatial objects, which is one of the most used operations in spatial analysis in GIS, is parallelized in a certain environment based on the classical uniform grid algorithm. An experiment is implemented to demonstrate the effectiveness of our strategy. The irregular distribution of spatial objects in the uniform grids is not considered in our parallel strategy, which is very important to load-balance, and this problem is what we will resolve in the future.

References

[1] Franklin Wm R, Chandrasekhar N, Kankanahili M, et al. (1988) Efficiency of uniform grids for intersection detection on serial and parallel machines [C]. Proceedings of CG International, Geneva, Switzerland.

[2] Wang Fangju (1993) A parallel intersection algorithm for vector polygon overlay [J]. IEEE Computer Graphics and Applications, 13(2): 74-81

[3] Michael Shamos, Dan Hoey (1976) Geometric intersection problems [C]. The 17th Ann. Conf. Found. Comp. New York, USA

[4] Bentley J, Ottmann T (1979) Algorithms for reporting and counting geometric intersection [J]. IEEE Trans. Computers C28: 643-647

[5] Bernard Chazelle, Herbert Edelsbrunner (1992) An optimal algorithm for intersecting line segments in the plane [J]. Journal of the ACM, 39(1): 1-54

[6] Mulmuley K (1988) A fast planar partition algorithm [C]. Proceedings of the 29th Annual IEEE Symposium Foundations of Computer Science, Washington, DC, USA

[7] Clarkson K L, Shor P W (1989) Applications of random sampling in computational geometry [J]. II. Discrete and Computational Geometry, 4(5): 387-421

[8] Ding Yuemin, Densham Paul J (1996) Spatial strategies for parallel spatial modeling [J]. Geographical Information Systems, 10(6): 669-698

Notes to Contributors

Contributions are welcomed on one of the following subjects or in related areas:

- GIS
- GPS
- RS
- Cartography
- Geodynamic
- Geo-surveying
- Photogrammetry
- Physical geo-surveying
- Engineering surveying
- Mapping apparatus
- Graphics

The main text should be preceded by the abstract of no more than 300 words, followed by key words. Full references should be listed in the order of the citations in the text under the heading “References”, guided by standard publication format. The name of the fund and project series number for articles of funded projects should also be given.

Contributions submitted for publication will be peer reviewed, i.e., the entire articles will be examined by qualified experts independent of the authors before publication.

The authors assume sole responsibility for their dissertations. Canned theses are not allowed. Once the paper is submitted, the editors are authorized to make necessary literate processing. Authors who receive no notice in one month after submission may contact us for inquiry.

Feel free to contact us.

Tel: +86-27-68778045
E-mail:gisis@whu.edu.cn
sjjs.2008@gmail.com