An Efficient Method of Map Generalization Using Topology Partitioning and Constraints Recognition

Hongtai Zhang[1,2] Jian Dai[1,2] Kuien Liu[1] Zhiming Ding[1] Huidan Liu[1]
[1]Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[2]University of Chinese Academy of Sciences, Beijing 100049, China
{hongtai,daijian,kuien,zhiming,huidan}@nfs.iscas.ac.cn

ABSTRACT
Map Generalization is one of the most fundamental technologies for modern digital maps. It can effectively reduce the storage space and fit to different applications according to their scale requirement. This paper presents an efficient solution for this problem that won the ACM SIGSPATIAL CUP 2014. Given the original geometries which are represented by sampling points sequence, this method divides the boundaries into many small segments based on their topological characteristics and constriants. It attempts to minimize the number of sampling points by simplifying the given map and constraining points. In addition, the method also employs many optimization techniques to reduce the total latency, like memory pool, parallel computing and string parsing. Experimental results on real datasets demonstrate the effectiveness and efficiency of the proposed method.

Categories and Subject Descriptors
H.2.8 [**Database Applications**]: Spatial databases and GIS

General Terms
Algorithms, Performance

Keywords
Topology Partitioning, Map Generalization, GIS, Cartography

1. INTRODUCTION
1.1 Background
Map generalization (i.e., map simplification) [2] is a technique that some key information is selected and kept on a map so that the generalized map can adapt to the scale of the display medium. By default, not all intricate geographical details are preserved, but the relationships between the geometries should be represented in the most faithful recognizable ways. Here, how to efficiently identify insignificant items and remove them (i.e., preserve the distinguishing items) is the main challenge [3-7].

1.2 Problem Definition

SIGSPATIAL'14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA

http://dx.doi.org/10.1145/2666310.2666423

Consider the MA state's map shown in Figure 1 that displays a set of state boundaries at a very detailed level. Some states share boundaries with other states and clearly the state boundaries do not overlap with each other. The red points represent the cities near the boundaries.

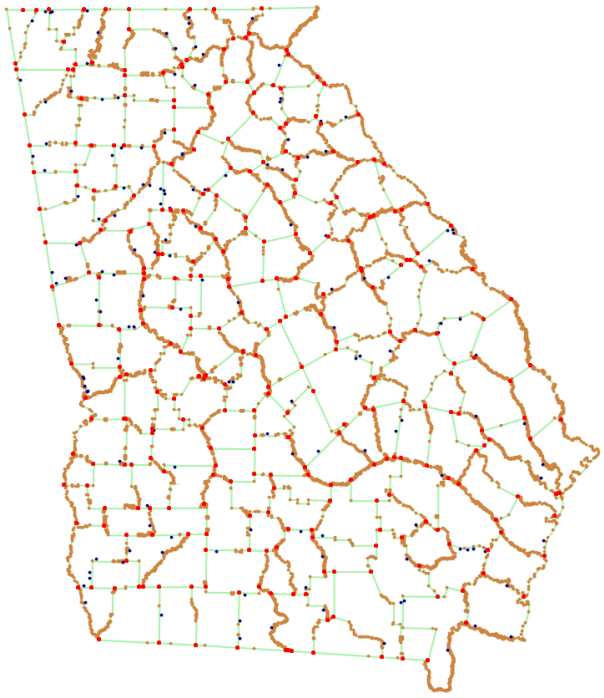
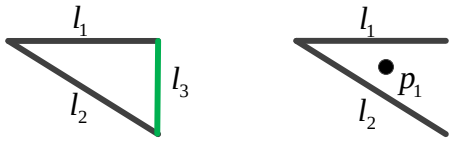


Figure 1. Visualization of MA state's Points and Boundaries

To facilitate map simplification, it is often desirable to break the state boundaries into different line geometries so that all shared boundaries are represented as unique line geometries. These lines are then simplified and connected back together to form the state boundaries. In this way, the state boundaries will still preserve the non-overlapping property they had before the boundary being simplified. However, this does not guarantee that the cities still maintain their relative position with respect to their state boundaries.

In this paper, we investigate the following problem [1]. If there're a set of linear geometries that bound polygonal regions and a set of constraining points. The objective is to simplify the linear geometries such that the relationship between the constraining points and linear geometries before and after the simplification does not change. In addition, the topological relationship between the original set of input linear geometries does not change after the simplification.



(a). without constraining point **(b). with constraining point**

Figure 2. Problem Definition

As shown in Figure 2(a), when there is no nearby constraining point, the line segments (i.e., l_1 and l_2) which comprise polygonal lines can be simplified to l_3. In contrast, as shown in Figure 2(b), if there is constraining point (i.e., p_1) in-between the line segments, we probably need further examination to determine if the points on the line segments can be simplified (i.e., eliminated). Specifically, in Figure 2(b), we cannot simplify l_1 and l_2 to l_3 anymore.

2. Proposed Approach

2.1 Preprocessing

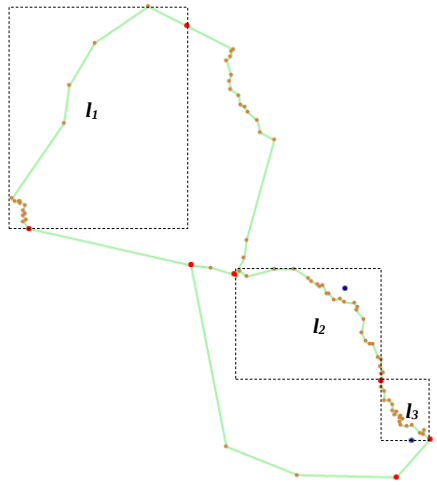


Figure 3. MBR of different line geometries

Via preprocessing, we aim to quickly eliminate those points that have no constraining points around. After finding those points, we can simplify the corresponding polygonal line by only keeping the starting point and the ending point of each such line segment.

In order to quickly find those points (i.e., polygonal line), we employ the Minimum Bounding Rectangle (MBR) to locate the line segments which have no internal constraining points.

Case 1. Specifically, a MBR is constructed by including all the points of a polygonal line. If there exists no constraining point inside the constructed MBR, then the corresponding polygonal line can be directly simplified by only keeping its starting point and its ending point, which is the mostly simplified polygonal line (i.e., line segment). For example, as shown in Figure 3, the polygonal line l_1 can be simplified by only keeping its starting point and its ending point.

Case 2. However, if there exist internal constraining points, we have to consider which points from the polygonal line can be simplified (i.e., eliminated). In this case, we employ the divide and conquer paradigm according to an observation that only the points near a constraining point which is inside the constructed MBR should be considered together. In contrast, if a point is far away from a constraining point, then to eliminate the point or not probably has nothing to do with the constraining point. For example, in Figure 3, the polygonal line l_3 cannot be simplified with the method used in case 1. We dub it case 2. In this case, we need to further examine the position relationship between the constraining points and the polygonal line. Sometimes, though there exist constraining points inside the MBR, the polygonal line can still be simplified by only keeping its starting point and ending point (e.g. the l_2 shown in the figure 3). Sometimes, however, some points of the polygonal line can be discarded while the other points (nearby the constraining points) cannot be discarded (e.g. the l_3 shown in the figure 3). Thus, we need to further apply fine-grained algorithms to determine.

As a conclusion for the preprocessing, we distinguish case 1 and case 2. For case 1, we can directly simplify the polygonal line. For case 2, we need to do further investigation.

2.2 Simplification

Simplification is the core step of our approach which is responsible for discarding the points of polygonal lines in case 2. The simplication is further composed of two substeps: partitioning and constraint recognition.

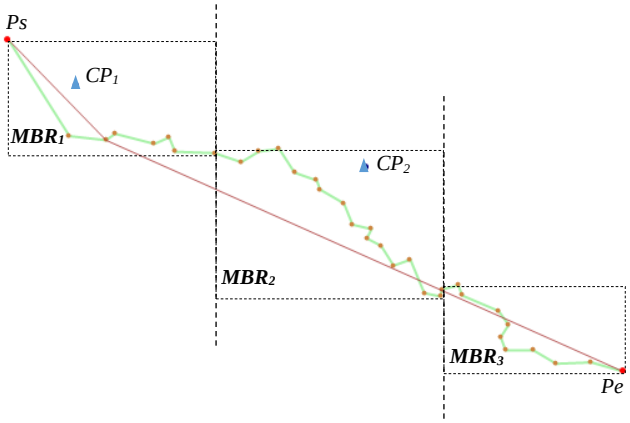


Figure 4. Partitioning

Partitioning. The partitioning step splits the original polygonal line into several sub polygonal lines where the number of sub polygonal lines is heuristically determined. We introduce this step to convert case 2 into case 1 mentioned in the preprocessing part. Because it is obvious that after partitioning the long polygonal line into several polygonal lines we can expect each sub polygonal line has less nearby constraining points (i.e., the corresponding MBR has less or none included constraining points). As illustrated in Figure 4, after partitioning the original polygonal line into three sub polygonal lines, the third polygonal line (included in the MBR_3) has no constraining point. If we split the original polygonal line into more sub polygonal lines, we should have less inside constraining points. However, the partitioning itself may become the most time-consuming step. To balance the time consumption, we propose that a long polygonal line (denoted by PL) should be splitted into m segments.

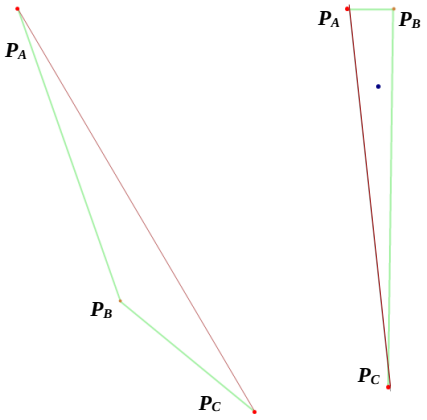
$$m = \left\lceil \frac{N_{sp}}{N_{cp}} \right\rceil$$

where N_{cp} denotes the number of constraining points included in the MBR determined by PL, and N_{sp} denotes the number of points

of *PL*. The motivation of this equation is straightforward, for m means the average number of points nearby each constraining point included in the MBR.

After computing *m*, the previous computed MBR with respect to the polygonal line is equally split into *m* cells alongside the x axis or y axis.

Constraint recognition. In this step, we aim to clearly detect if a point of the polygonal line can be discarded via recognizing if there exists a constraining point nearby that makes the point undeletable.



(a) case 3 (b) case 4

Figure 5. Constraint recognition

Figure 5 illustrates the basic idea behind the constraint recognition step. Overall, we employ a triangle determined by three consecutive points of the polygonal line to detect if there exists a constraint point inside the triangle.

Case 2-1. As shown in Figure 5(a), if no constraining point inside the triangle determined by the three consecutive points (i.e., P_A, P_B and P_C), then we can simplify the three points by removing the second point (i.e., only keep the starting and ending points).

Case 2-2. As shown in Figure 5(b), if there is any constraining point inside the triangle determined by the three consecutive points (i.e., P_A, P_B and P_C), then we have to keep the first point (i.e., P_A if we examine the points from P_A to P_B then P_C; otherwise, P_C if we examine the points from P_C to P_B then P_A). Please notice that the examination direction can be arbitrary for it will not influence the simplified result.

As aforementioned in the case 2-2, after examining the three consecutive points of a polygonal line, we move forward alongside the previously defined direction (e.g. $P_A \rightarrow P_B \rightarrow P_C$ or $P_C \rightarrow P_B \rightarrow P_A$) and examine the next three consecutive points, where each iteration only tries to remove one point (i.e. the second/middle point).

Specifically, the whole process can be described by the following poseducode. First (line 1-3), the total number of a polygonal line is computed and corresponding MBR is incrementally obtained. If there is a trival polygonal line, then we should return it directly (line 4-5). Otherwise, we further split it into multiple segments according to the huestically computed *m* (line 6-8) and examine each partition separately (line 9-13).

Algorithm 1 Simplification

Input: a line geometry *SPs* and constraining points set *CPs*
Output: a simplified line geometry
1. for each sampling point *p* of a polygonal line *SPs* do
2. MBR.extend(*p*);
3. *SPs*.num++;
4. if (2 == *SPs*.num || MBR don't contain any *p* in *CPs*)
5. return *SPs*.start and *SPs*.end;
6. else
7. $m=N_{sp}/N_{cp}$;
8. equally split the previously computed MBR into *m* ceil(*CPs*.num);
9. for every partition *P* in the *SPs*
10. if MBR don't contain any *p* in *CPs*
11. Retain Points Set RP_S.insert(*P*.start and *P*.end);
12. else RP_S.insert(every point in *P*)
13. constraint recognition (RP_S);
14. return RP_S;

However, some special cases (dentoed by ***S.C.***) require extra attention and hence corresponding detection algorithms are devised to deal with these special cases. Figure 6 and 7 demostrate the special cases we have found so far.

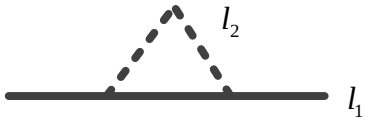


Figure 6. Special Case 1

S.C. 1. Since we should always keep the topological relationships between the original set of input linear geometries after the simplification, the two partially overlapping polygonal lines (i.e., l_1 and l_2) cannot be simplified into two points (i.e., only keep the starting and ending points) simoutaneously. Otherwise, the two lines will coincide which clearly violates the original topological relationships between them, since a part of l_2 locates above l_1. Therefore, in order to detect S.C.1, we have to examine the starting points and ending points of the polygonal lines which are included in the same MBR. In this case, we need to keep one more point which is not located both lines for l_2.

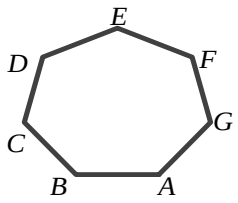


Figure 7. Special Case 2

S.C. 2. Likewise, if the original linear geometry is a circle-like structure, then we cannot simplify it as two points (i.e., only keep the starting and ending points). Otherwise, the corresponding island in the map disappears. We have to keep at least three points of the circle-like structure to make it look like an island. This special case can be easily detected by checking if the starting point is same as the ending point.

The special case detection procedure is invoked before the partitioning phase.

2.3 Some supplementary optimization

***IO optimization*:** In GIS, the geometries is usually representd in the GML format which is a modeling language. We optimize the document parsing with only one-time read from disk. In addition, we have implemented a simple but efficient *atof* function without considering variety of complex cases for floating number.

***Parallelized process*:** OpenMP is an implementation of multithreading, a method of parallelizing the computing task. In Map Generalization, the data loading and geometries preprocessing can be divided into several unrelated task. We set up 4 threads to finish those computing task according to the experiment environment. In addition, in order to minimize the cost of thread switching and communications, we design the multiprocess's calculation to be data independent.

***Memory pool*:** When we parse line and point geometry from input data files, it is costly to initialize and organize them into the data structure used by the algorithm. We design the memory pool to speed up the initialization process without requesting allocation of resources through the system every time.

3. Experiment

In this section we present the effectiveness of simplification algorithm (Topology Partitioning and Constraints Recognition) and other optimization strategies in terms of data size and running time, and finally we also verify the scalability of proposed method in different size of datasets.

In the experiments, we use training data sets 3 provided by ACM SIGSPATIAL Cup 2014 [1]. It contains the real line geometries data from boundary of the counties in GA state, USA and some point geometries which are used to constrain the line simplification process. The input lines geometries have a total of 8531 vertices and form 476 lines. Point geometries in this dataset is 151 in all. In order to test the actual performance of the algorithm, we expanded the original data set 50-fold to simulate application's data scale. Our implementation is written in C++ on Visual Studio express platform, experimental results were taken on a computer with Intel Core i7 Quad CPU 2600 3.1 GHz 64bit and 2 GB RAM.

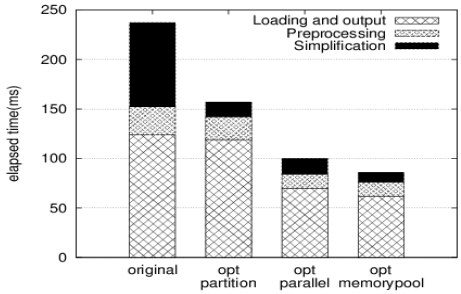


Figure 8. Effectiveness of the algorithm and optimization

To evaluate the method's performance, we set up multiple comparative experiments on the effect of the algorithm and various optimization strategies which are proposed in this paper. As shown in Figure 8, the Topology Partitioning algorithm reduce the simplification time to 17.64% as the orginal method using traversing the line geometries. However, loading and output comprises the most time-consuming parts of the overall process. We optimize the loading process via parallelization and the time reduces nearly 50%. Memory pool is also effective for loading and simplication, contributing toabout 30% improvement.

Figure 9 demonstrates the trend of different parts of this method with the growth of data size. It is clear that each part nearly increases linear with the data size's growth, loading and output process is the most significant increasing part.

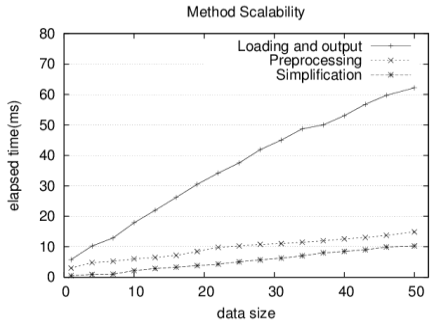


Figure 9. Method Scalability

4. CONCLUSIONS

In this paper, we propose an effective map generalization method using topology partitioning and constraints recognition. The experimental results demonstrate that the method achieves exciting effects compared to traversing line geometries.

5. ACKNOWLEDGMENTS

We would like to thank the ACM SIGSPATIAL Cup 2014 organizers for their valuable efforts during the competition. This work was supported by the China National 973 program 2014CB340301, the China National 863 program 2013AA01A603, and the National Science Foundation (NSF) China Grants 61202064 ,41371386 and 91324008.

6. REFERENCES

[1] ACM SIGSPATIAL Cup 2014

http://mypages.iit.edu/~xzhang22/GISCUP2014/index.php

[2] Buttenfield, B. P., & McMaster, R. B. (Eds.). (1991). *Map Generalization: making rules for knowledge representation*. New York: John Wiley and Sons.

[3] Urs Ramer, "An iterative procedure for the polygonal approximation of plane curves", Computer Graphics and Image Processing, 1(3), 244–256 (1972) doi:10.1016/S0146-664X(72)80017-0.

[4] David Douglas & Thomas Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature", The Canadian Cartographer 10(2), 112–122 (1973).

[5] Jones, C. B., Bundy, G. L., & Ware, M. J. (1995). Map generalization with a triangulated data structure. Cartography and Geographic Information Systems, 22(4), 317-331.

[6] Buttenfield, B. P., & McMaster, R. B. (Eds.). (1991). Map Generalization: Making rules for knowledge representation. New York: Longman Scientific & Technical.

[7] Liu, K., Li, Y., He, F., Xu, J., & Ding, Z. (2012, November). Effective map-matching on the most simplified road network. In Proceedings of the 20th International Conference on Advances in Geographic Information Systems (pp. 609-612).