A Fast Algorithm of Geometry Generalization

Yuwei Wang[†§] Danhuai Guo[†] Kuien Liu[‡] Yan Xiong[†]

[†]Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China
[§]University of Chinese Academy of Sciences, Beijing 100049, China
[‡]Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
{ahwangyuwei, guodanhuai}@cnic.cn, kuien@iscas.ac.cn, xiongyan@cnic.cn

ABSTRACT

Map generalization is commonly used in many GIS applications to produce maps with less detail so as to reduce the mapping complexity. Different from common simplifying strategies which simplify individual geometry objects separately, in this paper we consider the problem of generalizing the geometry objects under the topological constraints among the geometries and given constraining points. We propose a Cross-line algorithm to simplify the map while preserving the topological constraints. The proposed algorithm is extensively evaluated on five real map datasets and large synthetic datasets, and the results show that our proposed approach can greatly simplify the map with extremely high correctness and excellent efficiency.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial Databases and GIS

General Terms

Algorithms, Performance

Keywords

Geometry simplification, topological constraint

1. INTRODUCTION

In many mapping applications, a continuing challenge relates to rapidly mapping a large amount of geometry objects within a user's acceptable response time. A well known solution is map generalization [5, 1], which automatically and rapidly generalizes the map with less detail according to the map scale. The generalization reduces the complexity of the cartographic objects while preserving the general shape of the map.

Geometry generalization is a well known approach of map generalization, which reduces the complexity of the geometries while meeting an accuracy tolerance. Many algorithms for geometry generalization have been widely used such as the Douglas-Peucker algorithm (DP) [3] and the Visvalingam-Whyatt algorithm (VW) [6]. DP uses a divide-and-conquer strategy to simplify a polyline which is denoted as series of vertices $\langle p_1, p_2, \ldots, p_n \rangle$. Given a distance threshold ϵ, D-P removes the vertices of the polyline as many as possible while preserving that the deviation between the simplified polyline and the original polyline does not exceed ϵ [4]. Initially, DP tries to use the line segment p_1p_n as the final result. Then, DP examines whether the maximum distance of $P_i(1 \leq i \leq n)$ to the line segment p_1p_n exceeds ϵ. If it does not exceed ϵ, the process will terminate and return p_1p_n as the result. Otherwise, the vertex p_i will be retained, and the original polyline is divided at p_i as two sub-polylines $\langle p_1, p_2, \ldots, p_i \rangle$ and $\langle p_i, p_{i+1}, \ldots, p_n \rangle$. The simplification is recursively executed on the two sub-polylines separately. Instead of the maximum distance, VW considers the change of the area between the line and the simplified line after removing a vertex p_i, and the change is defined as the effective area of p_i. Given a number k of vertices to keep, VM iteratively removes the vertex which has minimal effective area, and recalculates the effective areas of the retained vertices. The process is repeated until the number of retained vertices reaches k.

These generalization algorithms including DP and VW, only consider the simplification of individual geometry and omit the relations between the geometries. This will cause some confusions due to the change of the original topological relations among the geometries, and hence mislead users. Considering the simplification of administrative boundaries [1], each administrative region contains a set of landmarks. If we simplify the boundaries independently, the memberships of some landmarks close to the boundaries w.r.t. the administrative regions may change. The change is not accepted in many critical applications.

In this paper, we consider the simplification of the geometries under the topological constraints. More specifically, given a large amount of geometries to be simplified and some Constraining Points (**CPs**), we try to simplify original geometries to a large extent while preserving the topological constraints among the geometries and the CPs. Because the polygon is a special form of the polyline, we could only emphasize the simplification of the polyline. We propose an efficient simplification algorithm named **Cross-line**. The algorithm simplifies the polylines in the horizontal and vertical directions sequentially. The basic idea is as follows: when simplifying a polyline w.r.t. a CP, we use a horizontal or vertical line across the CP to divide the polyline and then the consecutive intermediate vertices lie in the same side of the line can be removed. We generalize the process to simplification with multiple CPs and we argue that in most cases the topological relations remain unchanged. The main contributions presented in this paper are:

- We propose a Cross-line algorithm to simplify the geometries under the topological constraints of the geometries and the CPs, and further introduce some strategies to enhance the algorithm.
- We validate the effectiveness and efficiency of the proposed algorithm on five real map datasets and large synthetic datasets, and the results show that our algorithm has great accuracy and excellent efficiency.

The rest of the paper is organized as follows. Section 2 gives the description of the problem. Section 3 presents the proposed algorithm. Section 4 shows our experimental results and Section 5 summarizes this paper.

2. PROBLEM DESCRIPTION

Definition 1. A **polyline** L is a sequence of vertices $\langle P_1, P_2, \ldots, P_n\rangle$ where each vertex consists a coordinate pair $\langle x, y\rangle$.

Definition 2. A **simplified polyline** L' of a polyline $L = \langle P_1, P_2, \ldots, P_n\rangle$ is a subsequence of L, i.e. $L' = \langle P_{k_1}, \ldots, P_{k_{i-1}}, P_{k_i}, \ldots, P_{k_m}\rangle$, where $k_{i-1} < k_i$, $k_1 = 1$ and $k_m = n$.

Definition 3. The **topological relations** consists of **intersection relations** between polylines and **inclusion relations** between the CPs and the enclosed regions surrounded by some polylines. Here, we say that two polylines intersect with each other when one crosses the other except at the end vertices.

Given a set of CPs $\{CP_1, CP_2, \ldots, CP_m\}$ and a set of polylines $\{L_1, L_2, \ldots, L_k\}$ in which the polylines do not intersect with each other, our objective is to generate simplified polylines $\{L'_1, L'_2, \ldots, L'_k\}$ with as few vertices as possible while preserving topological relations. As the original polylines do not intersect, for the intersection relations we need guarantee that the simplified polylines do not intersect.

3. PROPOSED ALGORITHM: CROSS-LINE

3.1 Basic Process of Simplification Algorithm

First, we consider the inclusion relations involving a polyline $L = \langle P_1, P_2, \ldots, P_n\rangle$. Obviously, only the CPs which lie in the bounding box (BBOX) of L will influence the simplification of L. This scene can be classified into three cases: There are no (**Case 1**), one (**Case 2**), or more than one (**Case 3**) CPs in the BBOX.

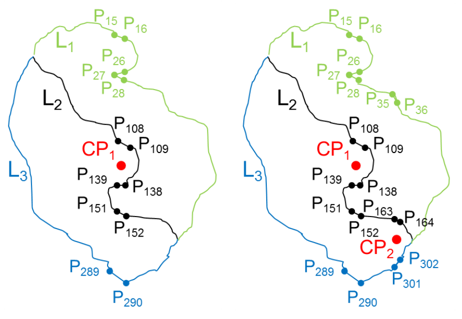
For Case 1, we can directly remove all intermediate vertices on the polyline. For Case 2, we design a two-stage strategy to remove all intermediate vertices w.r.t. the CP (i.e. CP_i). First, we utilize a straight line across CP_i (e.g. the vertical line $X = CP_i.x$ named **Y-Line**)to partition the space. Along L we sequentially look for the segments P_kP_{k+1} which intersects with the line. Considering two such adjacent segments P_kP_{k+1} and P_jP_{j+1}, we remove all the vertices i.e. $P_l(k+1 < l < j)$ between them, and obtain a simplified polyline L'. For further simplification, we use another line across CP_i (e.g. the horizontal line $Y = CP_i.y$ named **X-Line**) to execute the same process on L'. Theoretically, the line with any angle could be used. Here, we use X-Line and Y-Line because this leads to small computational cost for examining the intersection between the lines and the segments of L. Figure 1(a), where some intermediate vertices are highlighted, gives a sample of three polylines and a CP. Considering L_2, we first draw a Y-line across CP and get three intersecting segments $P_{108}P_{109}$, $P_{138}P_{139}$ and $P_{151}P_{152}$. We remove the intermediate vertices outside the segments and get L'_2 in Figure 2(a). Then, we simplify L'_2 to L''_2 in Figure 3(a) using the X-line. The inclusion relations are guaranteed by Lemma 1.

LEMMA 1. *Let $L = \langle P_1, P_2, \ldots, P_n\rangle$ be a polyline with BBOX B, and CP_i is a CP within B. The simplification of L using Cross-line will not change the inclusion relations between CP_i and the surrounded regions related to the L.*

PROOF. The lemma is easy to be verified. Consider the simplification of the polyline w.r.t. the first line AB across CP_i. Assume that P_kP_{k+1} and P_jP_{j+1} $(k+1 <= j)$ are two adjacent segments which both intersect with the line. The vertices $P_{k+2}, P_{k+3}, \cdots, P_j$ lie on the same side of L (Even if P_{k+1} or P_j is on the line, the Cross-line method acts the same). Removing the vertex $P_l(k+1 < l < j)$ will not change any inclusion relations w.r.t. CP_i. Further, utilizing another line CD across CP_i to simplify L' to L'' is a similar process and therefore is also valid. $\square$

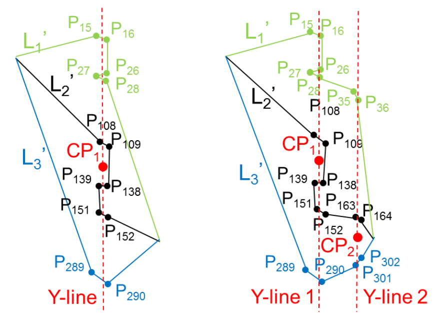
We extend the simplification of Case 2 to Case 3. Figure 1(b), Figure 2(b) and Figure 3(b) show an example with two CPs. Parallel X/Y-lines are drawn across the CPs, and are considered simultaneously, and segments intersecting with any line are retained. Note that we can not simplify a polyline w.r.t. the CPs one by one, because the simplification w.r.t. a CP might cause the topology change w.r.t. others.

For the intersection relations, we assert that in most cases the polylines would not intersect after simplifying all polylines using Cross-line. Figures 1∼3 illuminate the assertion. The assertion is also well validated in our experiments. A possibly better manner is to consider the vertices of other polylines as CPs when simplifying a polyline, but this will increase the complexity of the algorithm enormously.



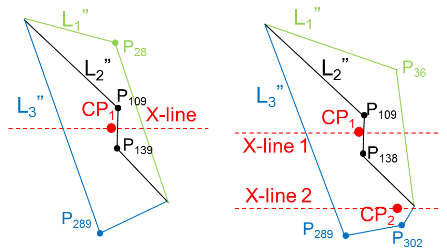
(a) Runtime (b) Memory requirement

Figure 1: The performance w.r.t. the extendible time threshold *ext*.



(a) Runtime (b) Memory requirement

Figure 2: The performance w.r.t. the extendible time threshold *ext*.



(a) Runtime (b) Memory requirement

Figure 3: The performance w.r.t. the extendible time threshold *ext*.

3.2 Enhancement of the Basic Cross-line

We design a heuristic strategy to decide the order of the lines used to improve the efficiency of Cross-line. We compare the width and height of the BBOX of L. If the value of the height is larger than that of the width, we call L a *high* line. Otherwise, we call L a *wide* line. If L is high, we use X-Line first. Otherwise, Y-line is first used. For example, $L2$ in Figure 1(a) is a high line. If we use X-line first, the L'_2 will has four vertices rather than eight vertices in 2(a). This would reduce the number of vertices that need to be processed in the next stage.

The basic process of Cross-line can handle most polylines, but there are still some special cases. The first concerns the polygon(i.e. the polyline with two identical terminals) whose BBOX containes no CP. When maximizing the simplification, the basic Cross-line would simplify the polygon as a vertex in this case. This can be avoided by retaining two intermediate vertices. The second case is that a simplified polyline becomes coincident with others. This occurs when two polylines are simplified to two coincident lines with two vertices (called **two-vertex polylines**). To avoid coincidence, we first record all raw polylines with only two vertices. When a two-vertex polyline is generated, we then examine whether it coincides with existing two-vertex polylines. If it does, we retain an intermediate vertex in it. Otherwise, we record it as a two-vertex polyline. For L_2 and L_3 in Figure 1(a), if CP_1 does not exist, we would simplify L_2 as two vertices and simplify L_3 as three vertices.

Algorithm 1: Cross-line

```
   Input: A polyline set $S_1 = \{L_1, L_2, \cdots, L_n\}$, a set of
          CPs $S_2 = \{CP_1, CP_2, \ldots, CP_m\}$
   Output: A set of simplified polylines $R$
1  $grid = buildGridIndex(S_2)$; // build a grid-based index for $S_2$
2  $S2 = collectTwoVertex(S_1)$; // record the information of existing two-vertex polylines
3  for $L_i \in CS$ do
4      $mbr = buildBBOX(L_i)$ ; // get BBOX of $L_i$
5      $S'_2 = pointsInBBOX(mbr, grid)$ ; // search for a subset of CPs in BBOX of $L_i$
6      if $isHigh(mbr)$ then
7          $L'_i = simpifyX(L_i, S'_2)$; // use X-lines first
8          $L''_i = simpifyY(L'_i, S'_2)$; // then use Y-lines
9      else
10         $L'_i = simpifyY(L_i, S'_2)$; // use Y-lines first
11         $L''_i = simpifyX(L'_i, S'_2)$; // then use X-lines
12     if $only2sameV(L''_i)$ then $retain2Middle(L'')$;
13     if $isTwoVertex(L''_i)$ then // $L''_i$ has 2 vertices
14         if $coincident(L''_i, S2)$ then
               // retain a middle vertex for $L''_i$
15             $retain1Middle(L''_i)$;
16         else
17             $updateS2(L''_i, S2)$;
18     $R.add(L''_i)$;
19 return $R$;
```

It is easy to see that the parallelization of Cross-line is straightforward. We can partition the set of polylines directly and then use general parallelization architecture such as Open Multi-Processing (OpenMP) [2]. For coincidence detection of the two-vertex polylines, we need to store existing simplified two-vertex polylines in a common memory. Each processing unit in the parallelization architecture needs to detect the coincidence and update the common memory.

3.3 Analysis of Cross-line

The process of Cross-line is shown in Algorithm 1. For fast search of the CPs in the BBOX of a polyline, we index the points using a grid-based structure (Line 1). $S2$ is the common structure to store the two-vertex polylines. In order to detect coincidence quickly, $S2$ is constructed as a hash table which hashes the union of two end vertices of a two-vertex polyline to a bucket. Lines 6~11 are the core steps of Cross-line which simplify a polyline using X-lines or Y-lines first. The coincidence handling is implemented in Lines 13~17.

For a polyline L_i, Cross-line detects its intersection with X/Y-lines for the CPs in S'_2. The average cost for all polylines in Lines 6~11 is $O(KM)$, where K is the average of $|S'_2|$ and M is the average length of L_i. The cost in Lines 12~17 is $O(1)$ due to the use of the hash table. Hence, the overall time complexity of Cross-line is $O(KN)$ where N is number of vertices in input. It is easy to see that the space complexity is $O(m + N)$ where m is the number of CPs.

4. EXPERIMENTS

We systematically evaluate Cross-line on five real bound-

Table 1: The description of the five experimental datasets and the results of Cross-line1 and Cross-line2

Dataset	Number of vertices	Number of polylines	Number of CPs	Cross-line1				Cross-line2			
				R (ms)	C (%)	A (%)	P	R (ms)	C (%)	A (%)	P
Dataset1	992	27	26	2.26	93.4	100	410.2	3.83	93.4	100	242.0
Dataset2	1564	46	127	3.79	92.9	93.4	353.8	5.74	91.5	100	249.3
Dataset3	8531	476	151	5.21	88.5	99.9	1447.8	7.95	88.4	100	948.9
Dataset4	28014	1353	356	9.11	90.0	99.9	2758.8	10.8	89.9	100	2333.9
Dataset5	28323	2331	1607	10.47	82.6	100	2235.2	11.53	82.6	100	2028.0

ary data of ACM SIGSPATIAL Cup 2014 [1] and large synthetic datasets expanded from the real datasets. The statistics of the datasets is shown in Columns 1∼4 of Table 1. In this paper, we use OpenMP with four threads to parallelize Cross-line. We compare two versions of Cross-line: Cross-line without (**Cross-line1**)/with (**Cross-line2**) coincidence detection as mentioned in Section 3.2. For Cross-line1, Lines 2 and 13∼17 in Algorithm 1 will be removed. The algorithm is implemented in C++ and executed on a machine with Intel(R) Core 2 Quad CPU Q8400@ 64bit 2.66 GHz.

We evaluate Cross-line in the following aspects: the runtime **R**, the simplification capacity **C** (the ratio of the total number of removed vertices to the total number of vertices), the accuracy **A** (the ratio of the total number of correctly removed vertices to the total number of removed vertices), and the overall performance **P** (the total number of removed vertices divided by the runtime). The statistics of Cross-line1 and Cross-line2 is shown in Table 1. We can see that both versions have extremely high simplification capacities and accuracies. The results show Cross-line can guarantee the topological relations in most cases. Comparing with Cross-line2, we can see that Cross-line1 introduces only extremely few errors and brings significantly better overall performance. This is because that the coincidence detection has a high computational cost even for $O(1)$ time complexity. Dataset 4 has the minimal ratio of the number of CPs to the number of vertices, so it generates a greater proportion of two-vertex polylines. Hence, Dataset 4 leads to the biggest difference of runtime for Cross-line1 and Cross-line2.

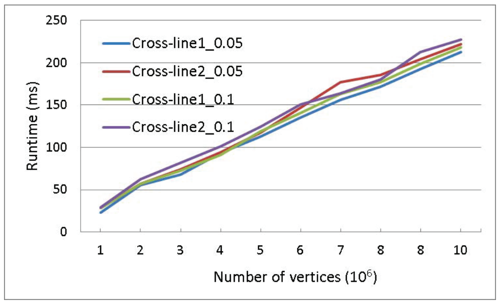


Figure 4: The runtime of Cross-line1 and Cross-line2 w.r.t. the data sizes (0.05 and 0.1 in the labels indicate the ratio of the number of CPs to the number of vertices).

We assess the efficiency of Cross-line on large synthetic datasets with various numbers of vertices and CPs. The results are shown in Figure 4. We can see that the runtime of both Cross-Line1 and Cross-Line2 increases linearly w.r.t. the number of vertices. When the number of CPs increases (i.e. varying the ratio of CPs to the number of vertices from 0.05 to 0.1 in Figure 4), the change of runtime is not obvious. There are two reasons for that. On one hand more CPs lead to more CPs in the bounding box of a polyline. On the other hand less vertices could be removed due to the greater chance of intersection with a X/Y-line for a segment.

5. CONCLUSIONS

In this paper, we propose a new algorithm for geometry simplification called Cross-line to simplify large number of geometries under the topological constraints of given constraining points and the geometry themselves. The experimental results show that the Cross-line algorithm has extremely high simplification capacity and accuracy. Meanwhile, Cross-line achieves very high computational efficiency, and this prompts us to adapt the algorithm for online map generalization.

6. ACKNOWLEDGMENTS

We would like to thank ACM SIGSPATIAL Cup 2014 organizers for the interesting competition. This work was supported by the National Natural Science Foundation of China (Nos. 41371386, 61202064 and 91324008), the National High Technology Research and Development Program of China (863 Program) (No. 2013AA01A603), and the Strategic Priority Research Program of the Chinese Academy of Sciences (No. XDA06010600).

7. REFERENCES

[1] ACM sigspatial cup 2014. Available from `http://mypages.iit.edu/~xzhang22/GISCUP2014/`. Accessed April 1, 2014.

[2] B. Chapman, G. Jost, and R. Van Der Pas. *Using OpenMP: portable shared memory parallel programming*, volume 10. MIT press, 2008.

[3] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[4] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *Proceedings of the VLDB Endowment*, 1(1):1068–1080, 2008.

[5] W. A. Mackaness, A. Ruas, and L. T. Sarjakoski. *Generalisation of geographic information: cartographic modelling and applications*. Elsevier, 2011.

[6] M. Visvalingam and J. Whyatt. Line generalisation by repeated elimination of points. *The Cartographic Journal*, 30(1):46–51, 1993.