Greedy Map Generalization by Iterative Point Removal

Yanzhe Chen
Shanghai Jiao Tong University
Shanghai, China
chenyanzhe@sjtu.edu.cn

Yin Wang
Facebook
Menlo Park, CA,
USA
yinwang@fb.com

Rong Chen Haibo Chen Binyu Zang
Shanghai Jiao Tong University
Shanghai, China
{rongchen,haibochen,byzang}
@sjtu.edu.cn

ABSTRACT

This paper describes a *map generalization* program we submitted to the ACM SIGSPATIAL Cup 2014. In this competition, the goal is to remove as many points in a set of polygonal lines as quickly as possible with respect to two constraints. The topological relationships among the lines must not change, and the relationships between a set of *control points* and the lines must not change. Inspired by Visvalingam-Whyatt Algorithm, we iteratively examine successive triplets along each line, and remove the middle point if no control point or point of other lines is in the associated triangle. Based on the features of the training datasets, we further introduce many optimization techniques to speed up the computation.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Applications—*Spatial databases and GIS*

General Terms

Algorithms, Experimentation, Performance

Keywords

map generalization, spatial index, spatial query

1. INTRODUCTION AND OVERVIEW

Geometry generalization is a well-known problem of selecting the information on a map in a way that adapts to the scale of the display medium of the map. It filters the unnecessary cartographic details while maintaining the map's purpose and actuality of the object being mapped. In SIGSPATIAL Cup 2014, we consider a special case of map generalization. The input is a set of polygonal lines that bound polygonal regions and a set of control points. The objective is to simplify the lines by removing its middle points yet preserving the topological relationships among the lines, as well as the relationships between control points and lines. The competition is evaluated by the number of points removed divided by the computation time, subject to the penalty on lines violating topological constraints and a required minimum number of points to be removed.

SIGSPATIAL '14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA
ACM 978-1-4503-3131-9/14/11. http://dx.doi.org/10.1145/2666310.2666422

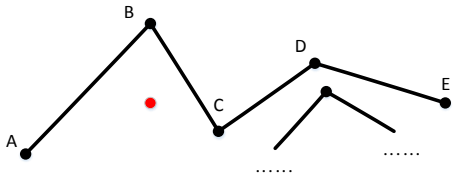

Figure 1: The middle point of a successive triplet can be removed if its associated triangle is empty, point C in this case.

There are two classical algorithms for map generalization. Ramer-Douglas-Peucker algorithm [1] recursively divides the polygonal line, and preserves the point which is furthest from the line segment between the two endpoints, if the distance exceeds a threshold. Visvalingam-Whyatt algorithm [4] iteratively eliminates a point with the smallest triangle formed by it and its two neighbor points. Neither of these algorithms takes into account the topological relationships among lines and between lines and control points. Therefore we cannot apply these algorithms directly to our map generalization problem.

Inspired by Visvalingam-Whyatt algorithm, however, we observe that it is safe to remove a point on a polygonal line if the triangle associated with the point and its two neighbors does not contain any control point or point from other lines. Figure 1 explains this idea. In this Figure, point B cannot be removed because its associated triangle ABC contains a red control point. Point D cannot be removed because triangle CDE contains a point of another line. Point C can be safely removed because triangle BCD is empty. Therefore we can examine all triangles associated with successive triplets of each polygonal line, and remove the middle point when the triangle is empty.

Multiple iterations of the above procedure can help eliminate more points. Figure 2 illustrates the idea. On the left side, there are two lines, ABC and DEF. We cannot remove B in the first iteration because E is inside triangle ABC, but it can be removed after we remove E. On the right side, we cannot remove B in the first iteration because control point P is inside triangle ABC, but C can be removed. The second iteration removes B.

Overall, our map generalization algorithm takes the fol-

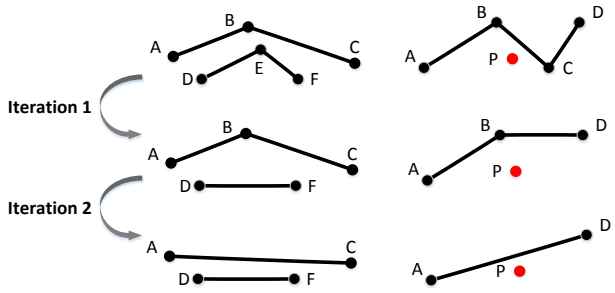

Figure 2: Iterations remove more points.

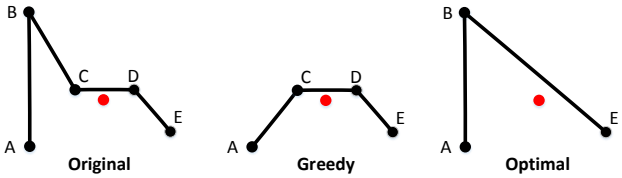

Figure 3: Our greedy algorithm is not optimal.

Table 1: Basic statistics of the provided datasets.

	lines	inner points	control points
Dataset1	27	992	26
Dataset2	46	1,564	127
Dataset3	476	8,531	151
Dataset4	1,353	28,014	356
Dataset5	2,331	28,323	1,607

lowing steps.

1. For each polygonal line, examine all successive triplets and remove the middle point if a triplet has no control point or point from other lines in its associated triangle.

2. Repeat Step 1 until no more point can be removed.

Our map generalization algorithm is greedy and it does not necessarily yield the minimally simplified solution. Figure 3 shows an example. If we remove point B first, both points C and D must remain to keep the red control point underneath the line. The optimal solution in this case is to keep just B. We choose this suboptimal greedy algorithm for a good balance between minimal simplification and computation speed.

Next we describe various optimization techniques in Section 2, and presents our evaluation results in Section 3. Section 4 concludes the paper.

2. OPTIMIZATION TECHNIQUES

In this section, we explain several optimization techniques employed by our program. Our optimizations are empirical in nature, based on the characteristics of the training datasets. Table 1 shows the datasets provided by the competition. The first three are training datasets given before the submission, and the last two are testing datasets released after the submission.

Our program finishes each of the datasets in at most tens of milliseconds after optimization. The time scale is too small for reliable measurements, i.e., operating system scheduling, disk hiccups can significantly affect the computation time. Sometimes even the overhead introduced by the measurement code can dominate the overall computation. Therefore, we created synthetic large datasets by shift-cloning the provided datasets [6]. We prefer cloning the provided datasets instead of generating random ones because we want to preserve the data characteristics for proper optimization. Since each dataset appears to have different characteristics, e.g., different ratio between lines and control points, we clone each dataset to about 200 MB, and use all of them in experiments. Table 2 shows the cloned datasets, where the name like "Dataset1_x4500" means Dataset1 shift-cloned 4,500 times.

Table 2: Large shift-cloned datasets.

	size of line file	size of point file
Dataset1_x4500	198 MB	21.5 MB
Dataset2_x3000	208 MB	70.3 MB
Dataset3_x500	204 MB	13.8 MB
Dataset4_x150	197 MB	9.79 MB
Dataset5_x150	221 MB	44.3 MB

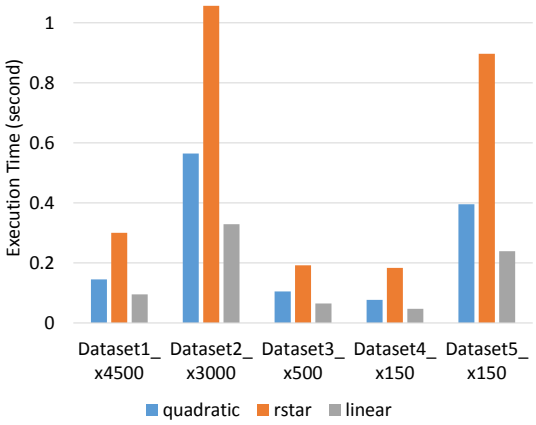

Figure 4: R-Tree building time (s) with different node splitting policies.

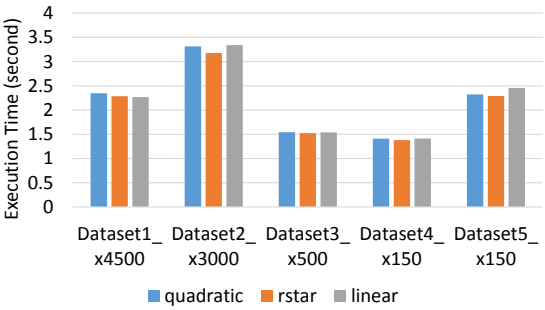

Figure 5: R-Tree querying time (s) with different node splitting policies.

2.1 Spatial Index

Checking whether there is any point in a triangle is the most time-consuming part of our algorithm. Spatial index is the key to achieving optimal performance. We employ R-Tree [2] in our implementation to index all points, and then get all points within the bounding box of a given triangle. For each point returned, we use `boost::geometry::within` function to check if it is inside the triangle.

Minimizing both *coverage* and *overlap* is crucial to the performance of R-tree. Different R-tree variants employ d-

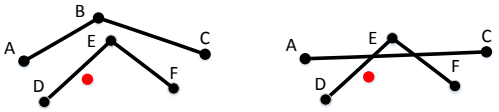

Figure 6: Incorrect result by ignoring inner points.

Table 3: Correctness rate of ignoring inner points.

	total lines	correct lines	correctness rate
Dataset1	27	27	100.0%
Dataset2	46	45	97.8%
Dataset3	476	473	99.4%
Dataset4	1,353	1,346	99.5%
Dataset5	2,331	2,317	99.4%

Table 4: Building and querying time including/excluding inner points in indexing.

Time (ms)	all points	excluding inner points
Dataset1	41	1
Dataset2	117	1
Dataset3	4,582	2
Dataset4	35,605	7
Dataset5	80,483	10

ifferent heuristics to split overflowing nodes, trying to minimize coverage and overlap [3]. We examined different node splitting policies implemented in the Boost library, quadratic, r-star, and linear, for both the tree building and querying time, shown in Figures 4 and 5. We can see that the node splitting policy has little effect on query performance, but makes substantial difference on building time. This is because there are lots of points to be indexed, but not many queries. We choose the linear splitting policy in our program.

2.2 No Indexing for Inner Points

Figure 1 shows that we need to index two sets of points for correctness, control points and points of all polygonal lines. Table 1 shows that points of lines are at least an order of magnitude more than control points. In addition, since we remove inner points of polygonal lines in each iteration, the index must be updated between iterations. Our experiments show that index building dominates the computation time.

We observe that since there are not many control points, the majority of inner points are removed. We can therefore ignore inner points in each iteration when checking whether any point is inside a triangle, i.e., indexing only control points and endpoints of all lines. Figure 2 shows an example where for the left part we can safely ignore point E and remove B since E will be removed later. This optimization is not always correct. Figure 6 shows an example where removing point B by ignoring E leads to incorrect simplification because E cannot be removed due to the red control point.

However, Table 3 shows that cases like Figure 6 are negligible. Table 4 further shows significant time saving by excluding inner points in indexing. Therefore we choose to exclude inner points in our R-tree index. Another crucial benefit of excluding inner points is that R-tree now becomes read-only, convenient for parallel processing, discussed in Section 2.4.

2.3 Fast Path

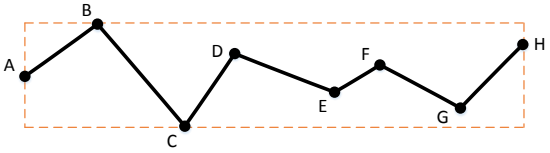

Figure 7: Fast path optimization.

Table 5: Fast path check success rate.

	Fast Path	Normal lines	Utilization
Dataset1	14	13	51.9%
Dataset2	19	27	41.3%
Dataset3	388	88	81.5%
Dataset4	1,181	172	87.3%
Dataset5	1,801	530	77.3%

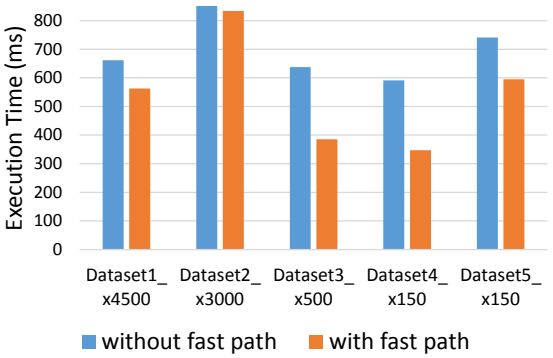

Figure 8: Time saving with fast path optimization at the point removal step.

Ignoring inner points significantly reduces the R-tree building time. Fast path optimization further reduces the number of R-tree queries. This is also based on the observation of fewer control points than line points. Based on Table 1, the number of control points is even less than the number of lines for most datasets. This implies that many polygonal lines are not adjacent to any control point and can be reduced to just one line segment. Fast path optimization achieves this. More specifically, we first build the bounding box for a line, and query R-tree to check if it is empty; see Figure 8. If so, we can remove all inner points and reduce the polygonal line to a line segment. Otherwise, we fall back and examine all triangles of successive triplets iteratively.

Table 5 shows that for most datasets fast path check succeeds in majority of the lines, and therefore substantially reducing the number of queries. Figure 8 shows the performance improvement at the point removal step.

2.4 Parallel Computation

Parallel computation is crucial for performance focused competitions like SIGSPATIAL Cup. Our program roughly consists of four stages, input, R-tree building, point removal, and output. For input and output, we apply the same parallel processing techniques used in our previous competitions [5, 6]. R-tree building is the only serial step. We apply a global lock to the tree, so each thread that processes the input needs to lock the tree before inserting a point. As discussed in Section 2.2, our R-tree is read only, so we can safely process multiple lines in parallel after the tree is built. We use the `parallel for` primitive from the OpenMP library in our implementation.

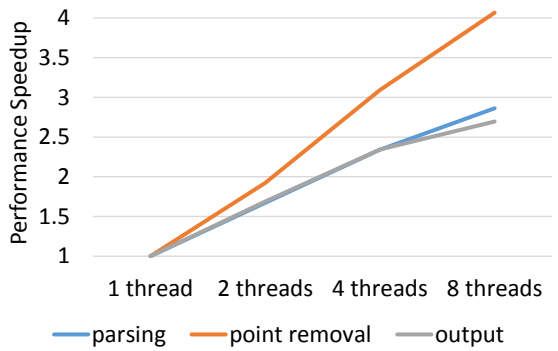

Figure 9: Parallel computation speedup for input parsing, point removal and output stages

Table 6: Computation time breakdown.

Time (ms)	input parsing	R-Tree building	point removal	output	all
Dataset1	0.00	0.33	0.33	2.00	1.50
Dataset2	0.75	0.25	0.75	4.00	5.67
Dataset3	2.33	0	1.00	88.67	93.33
Dataset4	5.67	0.67	4.00	15.33	24.00
Dataset5	6.00	2.67	4.67	10.57	22.33
Dataset1_x4500	615	130	581	341	1,735
Dataset2_x3000	720	361	816	473	2,358
Dataset3_x500	617	94	398	598	1,703
Dataset4_x150	582	69	339	523	1,539
Dataset5_x150	728	268	601	889	2,480

Figure 9 shows the parallel speedup for different computation stages, including input parsing, iterative point removal, and result output. We use the five large cloned datasets, and take the average of the normalized computation time to calculate the speedup factor.

2.5 Overlapping Lines

After simplification, sometimes two lines with the same endpoints both reduce to just one segment, and thus overlapping. In order to preserve the topological relationship, we must preserve at least one inner point of one line. Our solution is to store endpoints in a hash set whenever a line reduces to a segment. If the set has the endpoint pair already, a segment connecting the pair already exists. We therefore preserve the middle point of the current line so it does not overlap with the segment.

3. EVALUATION

We implemented our program in C++ using the Boost library for the R-Tree index and OpenMP library for parallelization. Experiments ran on a 64-bit Windows 8.1 machine, which has a 4-core 3.30 GHz Intel Xeon E3-1230 CPU and 8 GB memory. Table 6 shows the time breakdown and the overall time when running our program against both the real and our synthetic large datasets.

In this table, input parsing means the time needed to parse both polygonal line and control point files. R-Tree building is the serial step of constructing the index. Point removal is the iterative procedure discussed in Section 1, and output means writing the simplified lines to a file. For the five competition datasets, the time scale is too small to be reliably measured. The synthetic datasets show more consistent results. It can be seen from the table that input and output stages dominate the computation. Since the time measurement function introduces significant overhead, we remove the function when measuring the overall time for better accuracy, and therefore the overall time is not always the summation of all steps.

Table 7: Final results of our program and its variant that blindly removes all inner points of all lines

	time (ms)	points removed	grade
our program	311	58,221	187.2
variant	205	44,330	216.2

Just for fun, we submitted a variant of our program to the competition that simply removes all inner points of each line, or keeps one inner point if there is overlapping. Therefore, the running time of the variant is roughly the summation of the line parsing time and output time in Table 6. Table 7 shows the testing result reported by the program committee. It can be seen that the variant has much high grader than our full program due to its much faster computation time. Unfortunately it did not remove enough points as required by the competition, because incorrectly simplified lines do not count toward the total number of points removed.

4. CONCLUSION

In this paper, we first discussed the overall procedure of our iterative point removal algorithm for the map generalization problem. We then described several optimization techniques we developed based on the characteristics of the training dataset. Each optimization is supported by statistics of the dataset and experimental results show significant performance improvements. Our evaluation shows that our program processes each competition dataset in less than 100 ms, and our synthetic dataset with close to 5 million points in less than 2.5 s.

5. ACKNOWLEDGEMENTS

This work is supported in part by the Program for New Century Excellent Talents in University of Ministry of Education of China (No. ZXZY037003), Shanghai Science and Technology Development Funds (No. 12QA1401700), a foundation for the Author of National Excellent Doctoral Dissertation of PR China(No. TS0220103006), Doctoral Fund of Ministry of Education of China (No. 20130073120040), National Natural Science Foundation of China (No. 61303011 and 61402284) and Singapore NRF (CREATE E2S2).

6. REFERENCES

[1] 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*, 10(2):112–122, 1973.

[2] A. Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[3] A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis. *R-trees: Theory and Applications*. Springer, 2006.

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

[5] H. Wei, Y. Wang, G. Forman, Y. Zhu, and H. Guan. Fast viterbi map matching with tunable weight functions. In *ACM SIGSPATIAL*, 2012.

[6] T. Zhou, H. Wei, H. Zhang, Y. Wang, Y. Zhu, H. Guan, and H. Chen. Point-polygon topological relationship query using hierarchical indices. In *ACM SIGSPATIAL*, 2013.