International Cartographic Association
Association Cartographique Internationale

Simplification with Parallelism

Arthur van Goethem [a, *], Wouter Meulemans [a], Andreas Reimer [a], Bettina Speckmann [a]

[a] *TU Eindhoven, a.i.v.goethem@tue.nl, w.meulemans@tue.nl, a.w.reimer@tue.nl, b.speckmann@tue.nl*

* Corresponding author

Keywords: Generalization, simplification, algorithms

Abstract:

Map generalization requires various operators, such as selection, simplification and exaggeration to work together suitably. Algorithmically, many of these operators have been studied, with simplification and (label) selection being particularly fruitful areas. The interdependence between operators is complex, and results in highly complex algorithmic problems. While interdependence of operators is a well-known cause for said complexity, the pattern-generating interdependence of objects in a single operation is equally important: the simplification of geometric features should consider how other features of the same class are simplified. Patterns within the same object class are a key ingredient in cartographic communication: most prominently, isolines depend on visually interacting with each other in order to work at all (Imhof, 1965). Many other meso-structures such as settlement types or thematic elements like migration patterns have only visual interaction of individual objects as a signifier. Geometric similarity can come in many guises from visually “rhyming” via using similar curvatures and angles to direct repetition (Roberts, 2012). One approach to reach some of these effects are complex agent-based models. We explore if it is possible to already incorporate similarity constraints on the operator level, i.e., a single geometric algorithm. We begin with maintaining the parallelism across multiple objects that is present in the geometry. For example, alleys between buildings or height isolines cause different geometric objects to locally resemble or complement each other. Such local parallelism relations between objects should be preserved during generalization. Our argument complements the case made by Reimer and Meulemans (2011), who argue for parallelism as a driving factor in computing schematic representations for a single object, and the considerations by Haunert (2011), who presents methods for detecting symmetries in buildings for the purpose of simplification.

We present two simplification algorithms to promote parallelism by combining techniques from the Imai-Iri simplification algorithm (Imai & Iri, 1988) with dynamic programming. The first algorithm simplifies two curves concurrently, computing an optimal simplification considering parallelism between the two lines. The second algorithm iteratively simplifies one curve at a time and can easily be applied to more than two isolines, making it more directly applicable for larger datasets. Both approaches can relatively easily be modified to use other line simplification algorithms. We choose to stay with Imai-Iri, however, due to its well-understood mathematical attributes which allows us to stay as close as possible to provable results in this initial exploration of the topic.

Imai-Iri. Given a polyline P, the simplification algorithm by Imai and Iri (1988) determines for each pair of vertices whether the polyline between the vertices can be *shortcut*. The polyline can be shortcut between two vertices v and w if the straight segment from v to w is at most a distance ε away from the original polyline between vertex v and w. All the legal shortcuts together form a graph on the vertices of the polyline, called the *shortcut graph*. The algorithm then computes the shortest sequence of shortcuts that together cover the complete polyline by computing a shortest path in the shortcut graph. This method takes $O(n^2)$ time (Chan & Chin, 1996).

Defining parallelism. We base our definition of parallelism on the work of Reimer and Meulemans (2011). They define a parallelism score between two line segments, based on their angle and how well the “face each other”. Let α denote the angle between the two line segments, and L their bisector – that is, a line with angle $\alpha/2$ to both segments. We project both line segments onto L and intersect the corresponding intervals along L to obtain a value F. The parallelism score between these segments is then computed as $e^{-200\alpha^2} \cdot F$. The algorithms described below are agnostic to the exact parallelism measure used. Hence, they can easily be adapted to other parallelism measures.

Concurrent simplification with parallelism. We are given two polylines P and Q with m respectively n vertices, a parameter K bounding the total complexity of the output, and an error-margin ε. The two polylines should be simplified simultaneously such that the complexity (number of edges) of the polylines together is at most K, and the introduced error is at most ε, while promoting parallelism between the two polylines. Given ε we compute the shortcut graphs G_P and G_Q for P respectively Q. We then simultaneously find two paths through G_P and G_Q that are maximally parallel using the following two actions. First, one of the paths may be extended by a single shortcut. By definition such an extension does not improve the parallelism of the final result. Second, both paths may simultaneously be extended by a single shortcut each, improving the parallelism of the final result by the relative parallelism of the added two shortcuts. To find the combination of up to K shortcuts that maximizes the total parallelism we use dynamic programming. Define a dynamic-programming table where each entry $D[i, j, k]$ represents the simplification of P up to vertex i and the

simplification of Q up to vertex j, such that they together use exactly k edges and such that they maximize the total parallelism. The final answer is thus $\max_{1 \le k \le K} D[m, n, k]$, where ties are broken in favor of results with fewer edges.

We obtain the following recursive definition:

$$D[i,j,k] = \begin{cases} 0, & i = j = 1, k = 0 \\ \max \begin{Bmatrix} \max\limits_{(a,i) \in G_P} D[a,j,k-1] \\ \max\limits_{(b,j) \in G_Q} D[i,b,k-1] \\ \max\limits_{(a,i) \in G_P, (b,j) \in G_Q} D[a,b,k-2] + p(a,i,b,j) \end{Bmatrix}, & k \ge 1 \\ -\infty, & otherwise \end{cases}$$

Here, we assume that the maximum over an empty set is $-\infty$; and $p(a,i,b,j)$ represents the parallelism score between the shortcut from a to i in P and the shortcut from b to j in Q. The table has $O(n^2K)$ cells and computing one cell takes $O(n^2)$ time due to the third term of the second case. Hence, the DP runs in $O(n^4K)$ time, which dominates the quadratic time to initialize the shortcut graphs.

Our dynamic program (DP) optimizes what we refer to as "ordered parallelism": parallelism between two curves P' and Q' is defined by the sum of the parallelism scores of a sequence of pairs of edges (the shortcuts taken simultaneously). That is, parallelism is measured only for shortcuts occurring in the same order along both polylines. This is inherent in the DP as a result of the necessary "optimal" substructure. However, intuitively, such an order need not exist in all cases, as shown in Figure 1.

Iterative simplification with parallelism. The above DP has considerable running time, operates on only two polylines simultaneously, and is limited to ordered parallelism. To overcome these limitations, we propose an iterative alternative: we simplify the lines one by one, using earlier simplifications as context to determine parallelism. By repeatedly simplifying the lines in this manner, we converge on a representation with high parallelism.

The input to this algorithm is m polylines $P_1, \ldots, P_m$, with n vertices each (we assume for simplicity that all curves have the same complexity). Moreover, we assume we are given as input an $\varepsilon > 0$ that captures the error margin, a $K \le n$ that represents the maximal number of edges per simplified polyline (rather than over all lines), and an $X > 0$ indicating the number of iterations. We assume that the polylines are provided in "sorted order". That is, any two polylines P_i and P_{i+1} are also adjacent on the map and parallelism between these polylines should be maintained. We do not directly consider parallelism between polylines that are not adjacent in the order, even if the isolines are adjacent on the map.

First, we build the shortcut graph G_i and compute a simplification S_i for each polyline P_i in isolation, using Imai and Iri's algorithm. If the complexity exceeds K, no simplification with at most K edges exists within the error threshold.

After this initialization, we repeat the following procedure X times: for each polyline P_i, we consider each edge (u, v) in G_i and determine its parallelism score, denoted by $p(u, v)$, as the maximum over its parallelism score to each of the edges in S_{i-1} and S_{i+1}. We then update S_i to optimize for parallelism, using a DP for a single polyline akin to the one above. We define a DP table $T[v, k]$, that represents the highest parallelism score for a simplification of P_i up to vertex v using exactly k edges. The answer we seek is thus $\max_{1 \le k \le K} T[n, k]$, breaking ties in favor of results with fewer edges.

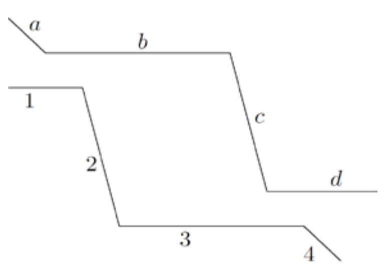

Fig. 1: *two curves with 4 edges. Though edge-pairs (b,3) and (2,c) are parallel, these pairs cannot occur in the same order along the curves; the dynamic programming method thus accounts for at most one of these pairs in terms of parallelism score.*

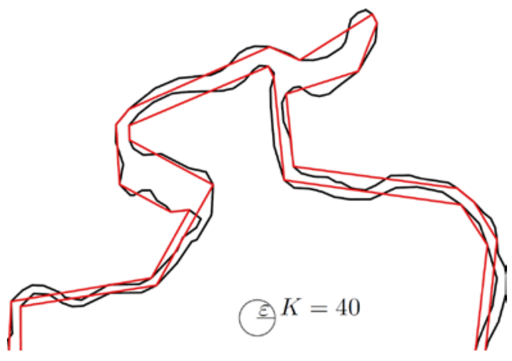

Fig. 2: *two curves with 53 (lower) and 87 (upper) edges in black; in red the result of the concurrent method, using K = 40 and ε as indicated by the radius of the circle.*

We obtain the following recursion:

$$T[v,k] = \begin{cases} 0, & v = 1, k = 0 \\ -\infty, & v = 1, k > 0 \text{ or } v > 1, k = 0 \\ \max\limits_{(u,v)\in G_i} T[u, k-1] + p(u,v)\,, & \text{otherwise} \end{cases}$$

Initialization using the Imai-Iri algorithm takes $O(m\,n^2)$ time. Each subsequent simplification costs $O(n^2\,K)$ time to update the weights, followed by $O(n^2\,K)$ time to compute the new simplification. One full iteration thus takes $O(m\,n^2\,K)$ time and the full algorithm thus amounts to $O(X\,m\,n^2\,K)$ time.

Discussion. We have described two approaches for simplification of polylines that incorporate parallelism. The first is slower but computes the "optimal" result for two polylines, the other a faster iterative approach that can handle multiple polylines simultaneously. Example results of our prototype implementations are shown in Figs. 2 and 3. We aim to investigate which of these approaches is more suitable by evaluating their performance in terms of running time, output complexity, and parallelism. For the iterative approach specifically, we aim to investigate how quickly this algorithm converges in practice.

There are still several other considerations that may be considered surrounding parallel simplification. First, there are some improvements to the described algorithms that would be desirable. The first algorithm can consider only ordered-parallelism measurements. It would be preferable if the approach could be extended to also take parallelism into account between shortcuts that do not occur ordered along both polylines. This appears to be hard to achieve without further increasing the runtime though. The second algorithm takes as input isolines in "sorted order". However, there are simple geographic features, like saddle points, that do not match such a linear ordering. An extension to these non-linear orderings appears a simple extension to the described algorithm.

Second, the algorithms as described consider parallelism between the complete isolines. It may be desirable to only consider parallelism between parts of the polylines. As an example, the extrusion in the outer isoline as displayed in Figs. 2 and 3 may preferably not be considered for parallelism with respect to the inner polyline. We are considering different techniques that automatically detect a mapping between two adjacent isolines to directly identify such extrusions and to exclude them from consideration regarding parallelism.

Third, both described approaches are vertex-restricted: the output is always a (not necessarily consecutive) subsequence of the input vertices. However, it may be desirable to be more flexible and to allow the introduction of new vertices for the simplification. Particularly, as a consequence of being vertex-restricted we may miss opportunities for parallelism that a smoothing operation could yield. In future work, we will investigate options for a smoothing operator or alternative approaches to introduce new vertices in the result.

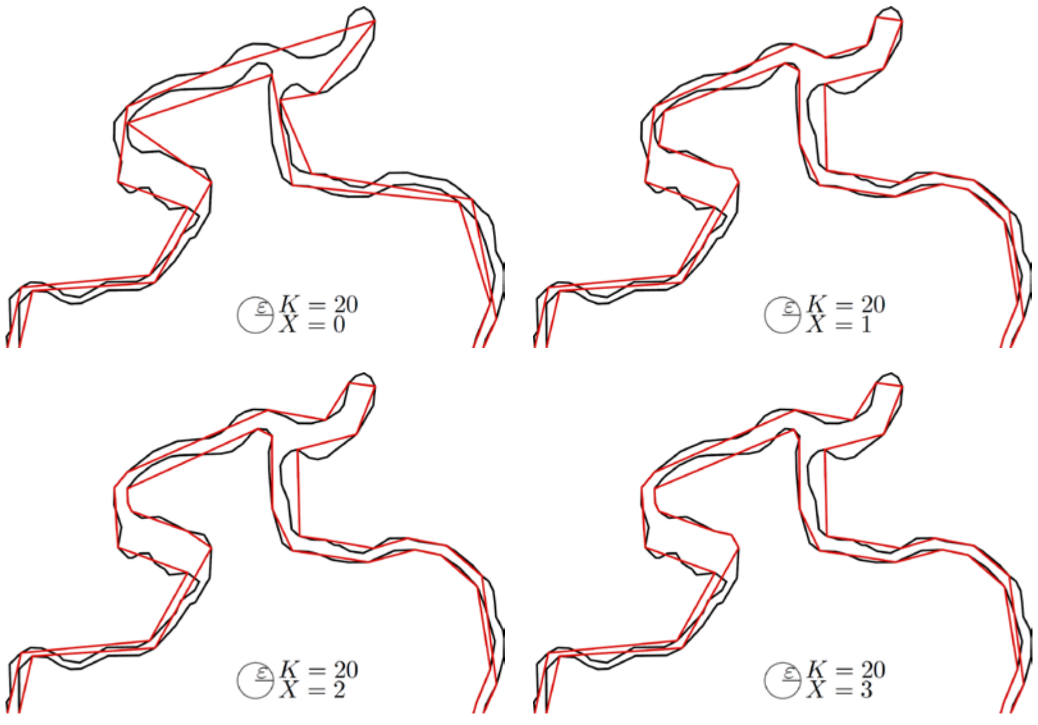

Fig. 3: *results of the iterative method using zero to three iterations are shown in red, for the same input as Fig 3. The same ε is used, and K is set to 20 to mimic K=40 as used by the concurrent method.*

References

W. S. Chan and F. Chin (1996). Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry & Applications*, 6(1): 59—77.

J.-H. Haunert (2011). Detecting symmetries in building footprints by string matching. *Advancing Geoinformation Science for a Changing World*, pages 319—336.

H. Imai and M. Iri (1988). Polygonal approximations of a curve - formulations and algorithms. In Godfried T. Toussaint, editor, *Computational Morphology: A Computational Geometric Approach to the Analysis of Form*.

E. Imhof (1965). Kartographische Geländedarstellung, Berlin.

A. Reimer and W. Meulemans (2011). Parallelity in chorematic territorial outlines. *Proceedings of the 14th Workshop on Generalisation and Multiple Representation.*

M. Roberts (2012). Underground Maps unravelled. Essex.