Gradual Simplification of Polylines

Stefan Funke[1] and Sabine Storandt[2]

1 **University of Stuttgart**
`funke@fmi.uni-stuttgart.de`
2 **University of Konstanz**
`storandt@inf.uni-konstanz.de`

Abstract

We propose the gradual polyline simplification problem, where a fine-grained succession of consistent simplifications of a given input polyline must be computed. We discuss different problem variants and present exact and approximate algorithms to solve them efficiently.

1 Introduction

Polyline simplification is the process of reducing the complexity of linear structures while ensuring that the output still resembles the main features of the input. There is a wide range of applications for polyline simplification, including data compression, noise reduction in trajectories, or visualization of linear structures on maps.

Formally, a polyline $L = p_1, p_2, \ldots, p_n$ is defined as a sequence of points and the induced straight line segments between consecutive points. In the polyline simplification problem, the input is a polyline, a distance measure d_X, and a threshold $\varepsilon > 0$. A line segment s_{ij} (also called shortcut) between polyline points p_i and $p_{j>i}$ is called *valid* if $d_X(s_{ij}, L[i,j]) \leq \varepsilon$, where $L[i,j]$ refers to the subpolyline of L from p_i to p_j. We also refer to $d_X(s_{ij}, L[i,j])$ as the shortcut error of s_{ij}. The goal of polyline simplification is to compute a minimum-sized path from p_1 to p_n that only uses valid shortcuts. The endpoints of these shortcuts define the simplified polyline $L' \subseteq L$. Typical similarity measures for polylines are the Hausdorff distance d_H and the Fréchet distance d_F. The polyline simplification problem can be solved optimally in $\mathcal{O}(n^2)$ for d_H [5] and in $\mathcal{O}(n^2 \log n)$ for d_F [6].

In [3], the problem of progressive simplification was introduced, motivated by the application of visualizing polylines on different levels of granularity e.g. in zoomable digital maps. Here, given a sequence of distance threshold values $\varepsilon_1 < \varepsilon_2 < \cdots < \varepsilon_k$, one needs to find a valid polyline simplification for ε_1, followed by a valid polyline simplification for ε_2, and so on, with the constraint that the simplification for ε_i contains of a subset of the points chosen for the simplification for ε_{i-1} for all $i > 1$ (also referred to as consistency). The optimization goal is to find a sequence of simplifications with the smallest number of shortcuts accumulated over all simplification levels. It was shown in [3] that the problem can be solved to optimality in $\mathcal{O}(n^3 k)$ for both d_H and d_F. Furthermore, a continuous version was discussed which can be solved in $\mathcal{O}(n^5)$.

Specifying a reasonable sequence of distance thresholds for progressive simplification is a non-trivial task. For example, it could easily happen that several consecutive ε values in the sequence induce the same simplification; thus only adding computational complexity but no further visual benefits. Furthermore, adding one ε to the sequence might impact all simplifications, as it might be beneficial to retain certain points in earlier simplifications to get smaller simplifications in later stages. The continuous version gets rid of the issue of needing to specify $\varepsilon_1 < \varepsilon_2 < \cdots < \varepsilon_k$ but at the cost of a significant increase in running time.

39th European Workshop on Computational Geometry, Barcelona, Spain, March 29–31, 2023.
This is an extended abstract of a presentation given at EuroCG'23. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

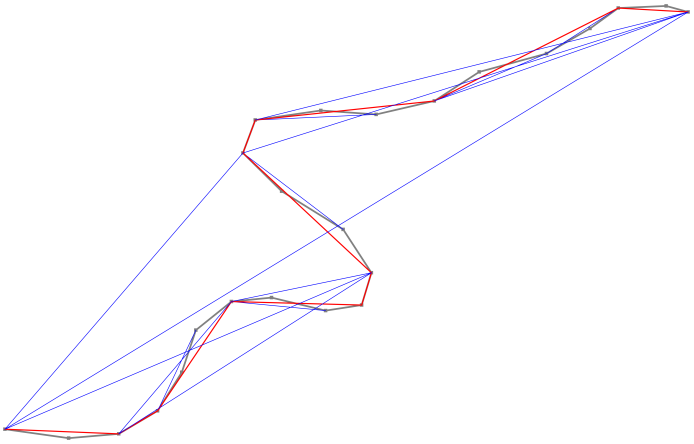

Figure 1 Example polyline (grey) and gradual simplification (blue and red shortcuts) with the optimal ordering for sum-error. The red polyline corresponds to the respective simplification of the input polyline after half the points have been shortcutted.

In this paper, we will introduce and discuss the notion of gradual line simplification as an alternative to progressive simplification.

2 The Gradual Line Simplification (GLS) Problem

While in progressive line simplification (PLS) the sequence of distance thresholds is provided as input and the goal is to find the smallest consistent simplification sequence that adheres to these thresholds, GLS takes an orthogonal view.

▶ **Definition 2.1** (Gradual Line Simplification)**.** Given a polyline $L = p_1, p_2, \ldots, p_n$, a gradual simplification corresponds to an ordering of the inner points $p_2, \ldots, p_{n-1}$ which are then shortcutted in the respective order. The goal is to find a contraction order that minimizes the maximum shortcut error (**max-error**) or the total sum of shortcut errors (**sum-error**).

Figure 1 illustrates the outcome of gradual line simplification on a small example instance. Consistency is not an issue for GLS as every contraction order yields a consistent simplification sequence. And compared to PLS, the produced simplification sequence is more fine-grained. However, GLS deviates from the wide-spread approach for polyline simplification where the distance threshold ε is provided. In this scenario, the validity of a shortcut is determined by deciding whether its error is smaller or larger than ε. In GLS, we need to compute the actual shortcut errors which is computationally more demanding. Nevertheless, we will show that contraction orders with high quality outputs can be computed efficiently.

3 Algorithms for GLS

We will propose exact and approximate algorithms to find good contraction orders for max-error GLS and sum-error GLS. The algorithms heavily rely on access to shortcut errors $d_X(s_{ij}, L[i,j])$. Under d_H, such errors can be computed in $\mathcal{O}(j-i) \subseteq O(n)$. Under d_F, using the algorithms by Alt and Godau [2], the error can be computed in $\mathcal{O}(n^2)$, or $\mathcal{O}(n \log n)$ if parametric search is applied. Recently, a data structure (FDS) proposed by Buchin et al. [4] allows to preprocess a given polyline in time and space $\mathcal{O}(n \cdot k^{3+\delta} + n^2)$ for $\delta > 0$ and then answer shortcut error queries in time $\mathcal{O}(\frac{n}{k} \log^2 n + \log^4 n)$ for some choice of $k \in [1, n]$.

3.1 Exact Algorithms

We first show that both the max-error and the sum-error problem can be solved to optimality in polytime. In particular, we design efficient dynamic programs (DP) that require $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space.

Let $\varepsilon(s_{ij})$ denote the shortcut error of s_{ij} with respect to $L[i,j]$. Further, let $\varepsilon_{\max}(s_{ij})$ denote the max-error of an optimal solution of the gradual simplification problem restricted to $L[i,j]$ (which automatically implies that s_{ij} is part of the solution). Similarly, $\varepsilon_{sum}(s_{ij})$ denotes the optimal sum-error of gradual polyline simplification for $L[i,j]$. Let p_k be the last point shortcutted before p_i and p_j, i.e. the point whose shortcutting resulted in the insertion of s_{ij}. Then the shortcuts (or original segments) s_{ik} and s_{kj} are part of the solution as well, and we have $\varepsilon_{max}(s_{ij}) = \max\{\varepsilon_{\max}(s_{ik}), \varepsilon_{\max}(s_{kj}), \varepsilon(s_{ij})\}$ and $\varepsilon_{sum}(s_{ij}) = \varepsilon_{sum}(s_{ik}) + \varepsilon_{sum}(s_{kj}) + \varepsilon(s_{ij})$. Based on these formulas, we can construct the solution set recursively starting with s_{1n}, which always has to be contained in any solution. However, we don't know the value of k if we go top-down. But this can easily be overcome by iterating over all possible k with $i < k < j$ and picking the minimum resulting max- or -sum-error. We can store already computed solutions for each s_{ij} in a look-up table to avoid redundant computations. This results in the following dynamic program: We allocate an $n \times n$ table and initially set all entries to 0. In cell c_{ij} with $i < j$, we store $\varepsilon_{\max}(s_{ij})$ or $\varepsilon_{sum}(s_{ij})$, respectively. The table cells are filled by using the above formulas. As we always need access to all shortcuts of smaller hop length to get the correct results, we consider the cells sorted increasingly by $j - i$.

▶ **Theorem 3.1.** *The DP approach solves max-error GLS and sum-error GLS in time $\mathcal{O}(n^3)$ under d_H and d_F, respectively, using quadratic space.*

Proof. The created table clearly has a space consumption of $\Theta(n^2)$. The running time is determined by the time needed to fill those $\Theta(n^2)$ cells. To get the correct cell value for c_{ij}, we first need to compute $\varepsilon(s_{ij})$ and then iterate over all values k between i and j and check cells c_{ik} and c_{kj}. The latter part can be accomplished in constant time per considered cell and hence takes $\mathcal{O}(n)$ in total. The computation of $\varepsilon(s_{ij})$ depends on the distance measure. For d_H, it takes time $\mathcal{O}(n)$. For the Frèchet distance, as discussed above, this would take $\mathcal{O}(n \log n)$ when using the parametric search technique. However, based on the FDS by Buchin et al. with a choice of $k = \sqrt{n}$, preprocessing the polyline and determining all potential shortcut errors can be accomplished in $o(n^3)$. Thus, we spend on average $O(n)$ time per cell for both d_H and d_F, which amounts to an overall running time of $\mathcal{O}(n^3)$. ◀

3.2 Approximation Algorithms

The cubic running time and the quadratic space consumption of the exact algorithm might be prohibitive in practice, especially when dealing with large inputs. Therefore, we next investigate approximation algorithms with better performance.

We first observe that the max-error problem variant is very easy to approximate for d_F based on a well-known lemma by Agarwal [1], rephrased below in our terminology:

▶ **Lemma 3.2.** *Given a polyline L, consider a shortcut s_{ij} for the subpolyline $L[i,j]$ with error ε. Then for any shortcut s_{ab} with $i \leq a < b \leq j$, its error under d_F is bounded by $d_F(s_{ab}, L[a,b]) \leq 2\varepsilon$.*

According to the lemma, the error of shortcut s_{1n} – which has to be contained in all gradual simplifications – is at least half the error of any other possible shortcut s_{ab} with $1 \leq a < b \leq n$. Thus, we get the following corollary.

▶ **Corollary 3.3.** *Any ordering is a 2-approximation for max-error GLS under d_F.*

Now let us turn to the sum-error problem variant. Note that the sum-error variant coincides with minimizing the average shortcut error, as the total number of shortcuts is $n-2$ for any ordering. In the following we present and analyze a greedy algorithm for sum-error GLS. The idea is to simply always choose the point next whose shortcutting will result in the currently smallest shortcut error.

▶ **Theorem 3.4.** *The greedy ordering is a 4-approximation for sum-error GLS under d_F.*

Proof. Let L be a polyline of size n. Let $S_1, \ldots, S_{n-2}$ be the shortcuts created by the greedy algorithm in their insertion order and let $\pi_1, \ldots, \pi_{n-2}$ denote the the points in L according to their contraction order. Further, let $S_1^*, \ldots, S_{n-2}^*$ be the shortcuts inserted based on an optimal point ordering. We use L_i or L_j^* to refer to the subpolylines shortcutted by S_i or S_j^*, respectively. Furthermore we use $\varepsilon(S)$ to denote the shortcut error of a shortcut S.

We construct an assignment of shortcuts inserted by the greedy algorithm to optimal shortcuts. In particular, we assign shortcut S_i to S_j^* if the following two conditions are met:

- At the moment before π_i is shortcutted by the greedy algorithm, there are at least three points in L_j^* that are not yet shortcutted, including π_i.
- Index j is the smallest index in the optimal ordering for which the above property holds.

Note that the assignment is well-defined, as we have $S_{n-2}^* = s_{1n}$ and the first condition is always true for S_{n-2}^* as we never shortcut its endpoints.

We now show that if S_i is assigned to S_j^* it holds $\varepsilon(S_i) \leq 2\varepsilon(S_j^*)$. This applies because if at least three points of L_j^* are not shortcutted before the shortcutting of point π_i, we know that shortcutting the middle of these three points would result in a shortcut for a subpolyline of L_j^*. According to Lemma 3.2, the error of any such shortcut is upper bounded by $2\varepsilon(S_j^*)$. As the greedy algorithm selects the next point to shortcut based on the minimum possible induced error at the current stage, we thus conclude that $\varepsilon(S_i) \leq 2\varepsilon(S_j^*)$ has to hold as well.

Let now c_j be the number of shortcuts S_i assigned to a particular shortcut S_j^* in the optimal solution. Then the greedy sum-error can be upper bound by $\sum_{i=1}^{n-2} \varepsilon(S_i) \leq \sum_{j=1}^{n-2} c_j \cdot 2\varepsilon(S_j^*)$.

To complete the proof, we will argue that $c_j \leq 2$ for all $j = 1, \ldots, n-2$. Assume now for contradiction that there exists a shortcut S_j^* to which we assigned at least three greedy shortcuts. Let the respective points that resulted in these shortcut insertions in that order be q_1, q_2, q_3. Based on the assignment criterion, we have $q_1, q_2, q_3 \in L_j^*$. Furthermore, there need to be three points that are not shortcutted yet at the point greedy considers q_3 in order for the respective shortcut to get assigned to S_j^*. Hence we have at least also $q_4, q_5 \in L_j^*$ that are shortcutted after q_3. Now we consider the point p^* that was shortcutted by S_j^* in the optimal solution. At that moment, there were only the points p_l, p^*, p_r left in L_j^*, where p_l is the left endpoint of S_j^* and p_r the right one. Hence the shortcuts (or original line segments) $S_l = (p_l, p^*)$ and $S_r = (p^*, p_r)$ exist in the optimal solution. If those are shortcuts, their index needs to be smaller than j, as they were constructed before S_j^*. Now if we have q_1, q_2, q_3, q_4, q_5 in L_j^*, at least three of them have to be contained in either the subpolyline belonging to S_l or S_r. W.l.o.g. assume it is S_l. That automatically implies that S_l is indeed a shortcut and not an original segment. Now if we shortcut the q_i with smallest index i in S_l (which implies $i \leq 3$), we assign S_i to S_l, as S_l has a smaller index than S_j^*, and there are at least three not yet shortcutted points in the respective subpolyline including q_i. This contradicts our claim that S_i is assigned to S_j^*. We therefore conclude that $c_j \leq 2$ holds. Accordingly, we get $\sum_{j=1}^{n-2} c_j \cdot 2\varepsilon(S_j^*) \leq \sum_{j=1}^{n-2} 2 \cdot 2\varepsilon(S_j^*) = 4\sum_{j=1}^{n-2} \varepsilon(S_j^*) = 4 \cdot OPT$. ◀

Thus, a greedy ordering provides simultaneously a 2-approximation for max-error and a 4-approximation for sum-error (or average-error).

Regarding the running time, we observe that the greedy algorithm only needs to compute linearly many shortcut errors instead of the quadratically many required by the exact algorithm: There are the initial errors of shortcuts s_{ii+1} for $i = 1, \ldots, n-1$ which can be computed in constant time each. Then, after selecting the next point to contract, only the values for its two former neighbors need to be updated. As there are $n-2$ contractions overall, the total number of non-trivial error computations is $2n-4$. If we use the standard approach by Alt and Godau, then computing the errors would take $\mathcal{O}(n^3)$ in total and thus the running time would not be better than for the exact approach. If we use parameteric search, then the time decreases to $\mathcal{O}(n^2 \log n)$. For these two variants the space consumption is linear. However, we can again exploit the FDS by Buchin et al. to achieve further speed-up (while accepting an increased space consumption).

▶ **Lemma 3.5.** *The greedy algorithm can be implemented to run in $\mathcal{O}(n^2)$ time and space.*

Proof. Recall that FDS requires a preprocessing time of $\mathcal{O}(n \cdot k^{3+\delta} + n^2)$ and has a query time of $\mathcal{O}(\frac{n}{k} \log^2 n + \log^4 n)$ for some $k \in [1, n]$. As we need to issue $\mathcal{O}(n)$ shortcut error queries, the total time of the greedy algorithm based on the DS can be expressed as $\mathcal{O}(n \cdot k^{3+\delta} + n^2 + \frac{n^2}{k} \log^2 n + n \log^4 n)$. If we choose k to be slightly smaller than $n^{1/3}$, e.g. $k = n^{(3-\delta)/9}$, then all summands are in $\mathcal{O}(n^2)$. Selecting $n-2$ times the one with minimum error among the current $\mathcal{O}(n)$ shortcut candidates can also be accomplished in $\mathcal{O}(n^2)$. ◀

4 Future Work

For the Hausdorff distance, the greedy algorithm could be implemented potentially even faster when using suitable data structures. But it is unclear whether this would also provide a constant factor approximation for gradual line simplification. Another direction for future work would be the investigation of different objective functions, as e.g. the sum of squared shortcut errors, or the sum over the shortcut errors of each individual simplification.

References

1 Pankaj K Agarwal, Sariel Har-Peled, Nabil H Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3), 2005.

2 Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 1995.

3 Kevin Buchin, Maximilian Konzack, and Wim Reddingius. Progressive simplification of polygonal curves. *Computational Geometry*, 2020.

4 Maike Buchin, Ivor van der Hoog, Tim Ophelders, Lena Schlipf, Rodrigo I. Silveira, and Frank Staals. Efficient Fréchet Distance Queries for Segments. In *30th Annual European Symposium on Algorithms (ESA 2022)*, 2022. doi:10.4230/LIPIcs.ESA.2022.29.

5 Wing Shiu Chan and Francis Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry & Applications*, 1996.

6 Sabine Storandt and Johannes Zink. Polyline simplification under the local Fréchet distance has subcubic complexity in 2D. *arXiv preprint arXiv:2201.01344*, 2022.