Simplification of Polyline Bundles

Joachim Spoerhase[1], Sabine Storandt[2], and Johannes Zink[3]

1 **Aalto University, Finland**
`joachim.spoerhase@aalto.fi`
2 **AG Algorithmik, Universität Konstanz.**
`storandt@inf.uni-konstanz.de`
3 **Lehrstuhl für Informatik I, Universität Würzburg.**
`zink@informatik.uni-wuerzburg.de`

Abstract

We propose a generalization to the well-known problem of polyline simplification in two variants. We are given, instead of a single polyline, a set of polylines possibly sharing some edges and vertices. We show that this more general problem is *NP*-hard by a reduction from Max-2-SAT. On the positive side, we show fixed-parameter tractability in the number of shared vertices.

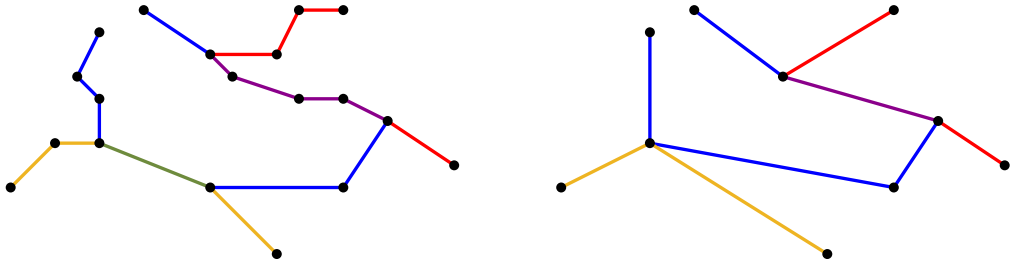
1 Introduction

Visualization of geographical information is a task of high practical relevance, e.g., for the creation of online maps. Such maps are most helpful if the information is neatly displayed and can be grasped quickly and unambiguously. This means that the full data often needs to be filtered and abstracted. Many important elements in maps like borders, streets, rivers, or trajectories are displayed as polylines (also known as polygonal chains). For such a polyline, a simplification is supposed to be as sparse as possible and as close to the original as necessary. A simplified polyline is constructed by a subset of vertices of the original polyline such that the (local) distance to the original polyline does not exceed a specifiable value according to a given distance measure, e.g., the Hausdorff distance [4] or the Fréchet distance [1]. The first such algorithm, which is still of high practical importance, was proposed by Ramer [7] and by Douglas and Peucker [3]. Hershberger and Snoeyink [5] proposed an implementation of this algorithm that runs in $O(n \log n)$ time, where n is the number of vertices in the polyline. It is a heuristic algorithm as it does not guarantee optimality (or something close to it) in terms of retained vertices. An optimal algorithm in this sense was first proposed by Imai and Iri [6]. Chan and Chin [2] improved the running time of this algorithm to $O(n^2)$.

From a Single Polyline to a Bundle of Polylines

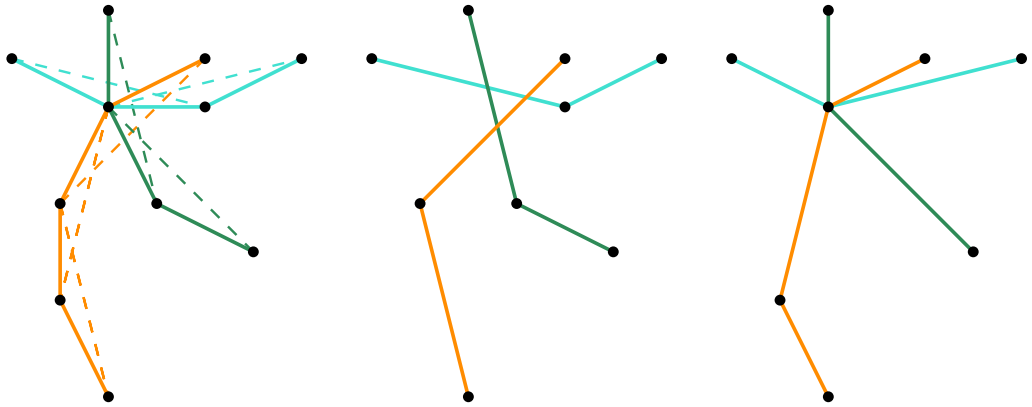
On a map, there are usually multiple polylines to display. Such polylines may share vertices and edges sectionwise. For example, when considering (GPS) trajectories like car-routes, different trajectories may partially share edges and vertices when cars have been on the same roads. Another example is a schematic map of a public transport network. Bus lines are the polylines and the vertices are the stations. In the city center, there are many different bus lines at the same stations that fan out when going to the outer districts, where they possibly share stations with further different bus lines. One might consider simplifying the polylines of a bundle independently. This has some drawbacks, though. On the one hand, the total complexity might even increase when the shared parts are simplified in many different ways. On the other hand, it might suggest a misleading picture when we remove common edges and vertices of some polylines, but not of all. The viewer might get the wrong impression that the one route has taken some street or passed through some area and the other has not, while in reality both took the same route in this place. E.g., if there is only one way to

35th European Workshop on Computational Geometry, Utrecht, The Netherlands, March 18–20, 2019.
This is an extended abstract of a presentation given at EuroCG'19. 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.



(a) initial bundle of polylines **(b)** simplified bundle of polylines

Figure 1 Example of a bundle of three polylines before and after the simplification.



(a) initial bundle with shortcuts **(b)** optimal for Min-Edges **(c)** optimal for Min-Vertices

Figure 2 Example of three polylines, where the goals Min-Edges and Min-Vertices differ.

pass through some point, then the simplifications of all polylines going through this point should still share the corresponding vertex or edge if it is kept. Therefore, we require that a vertex in a simplification of a bundle of polylines is either kept for all polylines containing it or discarded in all polylines. In Figure 1, we give an example of a simplification of a bundle of polylines. Natural minimization goals are to minimize either the total number of vertices (Min-Vertices) or the total number of line segments, i.e., edges (Min-Edges). Both goals generalize the previously described minimization problem for a single polyline. However, they may differ from each other like in Figure 2. In this extended abstract, we focus on Polyline-Bundle-Simplification with the goal Min-Edges to be formalized next. With small adaptions, our results also hold for the goal Min-Vertices.

2 Problem Definition

In an instance of the problem Polyline-Bundle-Simplification with goal Min-Edges, we are given a set $V = \{v_1, \ldots, v_n\}$ of n points in the plane, and a set $\mathcal{L} = \{L_1, \ldots, L_\ell\}$ of ℓ polygonal chains $L_i = (s_i, \ldots, t_i)$ represented as lists of vertices from V, as well as a distance parameter ϵ referring to a distance measure d (e.g., Hausdorff distance). The goal is to obtain a subset $V^* \subseteq V$ of the points, such that for each L_i its induced simplification S_i, which is $L_i \cap V^*$ while preserving the order of vertices,

- contains the start and the end vertex of L_i, i.e., $s_i, t_i \in S_i$, and
- has at most a distance of ϵ to L_i, i.e., for each line segment (a, b) of S_i and the

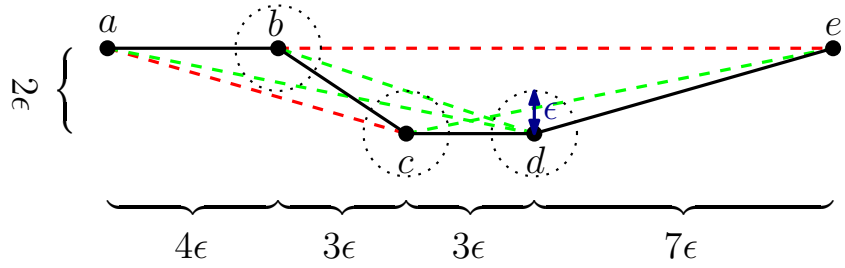


Figure 3 The literal-gadget depicted in black with valid (invalid) shortcuts in green (red).

corresponding sub-polyline of L_i from a to b, abbreviated by $L_i[a, \dots, b]$, we have $d((a, b), L_i[a, \cdots, b]) \leq \epsilon$,

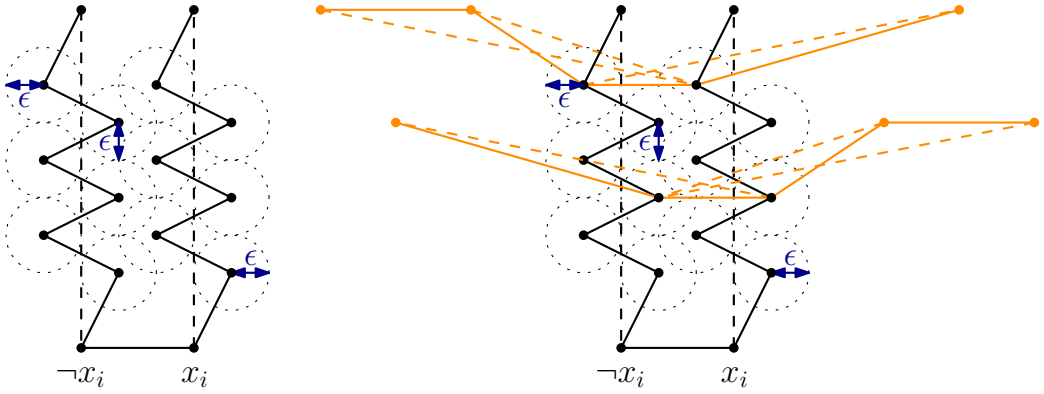
and the total number of edges induced by V^* is minimized. Here an edge (v, w) is induced by V^* if there exists at least one polyline L_i with $L_i[v, \dots, w] \cap V^* = \{v, w\}$. Note that we do not count the edge multiple times if there are multiple such polylines.

3 NP-Hardness

The problem Polyline-Bundle-Simplification with goal Min-Edges is *NP*-hard even for only two polylines (hence not *FPT* in ℓ unless $P = NP$). We show this by a reduction from Max-2-SAT, which is known to be *NP*-complete. In this reduction, we model a given 2-SAT formula by an instance of Polyline-Bundle-Simplification with goal Min-Edges using two polylines. Our reduction uses three types of gadgets, which allow some shortcuts inside the gadgets: *literal-gadgets*, *variable-synchronization-gadgets*, and *clause-synchronization-gadgets*. We obtain the first polyline by connecting all literal-gadgets such that no new shortcuts are possible. The second polyline is the connection of all variable-synchronization-gadgets and all clause-synchronization-gadgets such that no new shortcuts are possible. Next, we specify the three gadgets of our reduction.

Literal-Gadget. We model each literal of each clause by a *literal-gadget*. In Figure 3, a literal-gadget for modeling a positive literal x is depicted. For negative literals, it is the same but mirrored horizontally. It consists of five serially connected vertices (drawn in black). Valid shortcuts are dashed in green, invalid shortcuts in red. The vertices a and e cannot be skipped and the inner vertices b, c, and d are shared with the second polyline. There are three mutually exclusive shortcuts: skipping b and c, skipping c, and skipping d. Skipping c (together with or without b) or d is always possible and corresponds to the truth assignment of this clause. Since the number of edges is minimized, the shortcut that skips b and c will be chosen whenever possible (in compliance with the variable-synchronization-gadget, with which c and d are shared, and the clause-synchronization-gadget, with which b is shared). The interpretation is as follows: if c is skipped, x is set to true; if d is skipped, x is set to false; if b is skipped, this literal satisfies its clause. So b is the "critical" vertex indicating that a clause is satisfied. In a literal-gadget for a negative literal, b is between d and e, and not between a and c. Clearly, all vertices lie on a grid point of a grid with square length ϵ.

Variable-Synchronization-Gadget. For each variable, we use a *variable-synchronization-gadget* to enforce a consistent truth assignment for a variable x_i. In Figure 4, a variable-synchronization-gadget for synchronizing six literal-gadgets is depicted. Shortcuts are depicted as dashed segments in the color of its polyline. The number of vertices in the



(a) The gadget alone.

(b) The combination of a variable-synchronization-gadget (black) and literal-gadgets (orange). Only 2 of 6 literal-gadgets are drawn here.

Figure 4 The variable-synchronization-gadget.

section going zigzag corresponds to the number of occurrences of this variable—regardless of positive or negative. Except for the two vertices on the top and the two vertices on the bottom of the gadget, all vertices are shared with the literal gadgets—each two vertices with the same y-coordinate are part of the same literal gadget (the vertices c and d in Figure 3). There are only two shortcuts: skipping all shared left vertices and skipping all shared right vertices. The interpretation is as follows: if we skip the left vertices and keep the right vertices, x_i is set to true and the other way round x_i is set to false. An inconsistent assignment is not possible: we cannot take both shortcuts, since we cannot skip both lower vertices in a literal gadget. Taking none of these shortcuts would violate the minimality, since consistently skipping the same lower vertex in the literal gadgets is always possible (otherwise we cannot take any shortcut of the concerned literal-gadgets). Again, all vertices lie on a grid point of a grid with square length ϵ.

Clause-Synchronization-Gadget. We use a *clause-synchronization-gadget* for each clause with two literals. Its purpose is to reward satisfied clauses uniformly, i.e., it prohibits "double" satisfied clauses from being rewarded better than "once" satisfied clauses. In Figure 5, a clause-synchronization-gadget is depicted in black. It consists of four serially connected vertices, which connect two literal-gadgets (gray color) that correspond to two literals of the same clause. The inner vertices b_1 and b_2 are shared with the two b-vertices of these literal-gadgets (compare with Figure 3). Valid shortcuts are dashed in green, invalid shortcuts in red. There are two mutually exclusive shortcuts: skipping b_1 and skipping b_2. If one of them is used, then the b-vertex of one literal-gadget is skipped. This is only possible when the assigned truth value satisfies the corresponding literal, which in turn satisfies the corresponding clause. We can say: for each satisfied clause, we get the reward of reducing the total number of edges by two when we skip such a b-vertex (the one edge in the literal-gadget, the other edge in the clause-synchronization-gadget), which we cannot remove otherwise. Since there is no shortcut from s to t, it is not possible to skip both b-vertices corresponding to the same clause and, therefore, also not possible to get a greater reward if both literals of the same clause are set to true. Since we minimize the number of remaining edges, as many clauses as possible are satisfied this way because if at least one of the corresponding literal-gadgets is set true, we clearly can also skip the b-vertex of this literal-gadget. Hence, only if none of the two b-vertices is skipped, the corresponding clause remains unsatisfied.

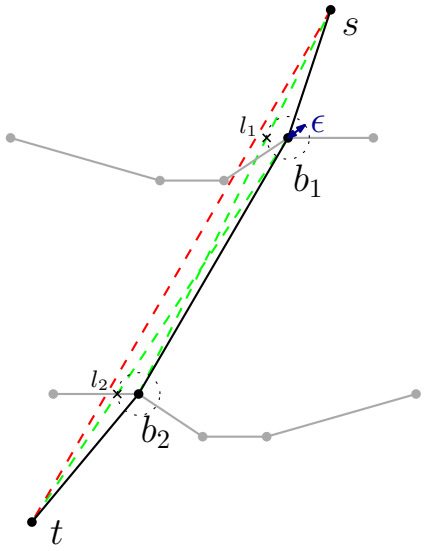


Figure 5 The clause-synchronization-gadget.

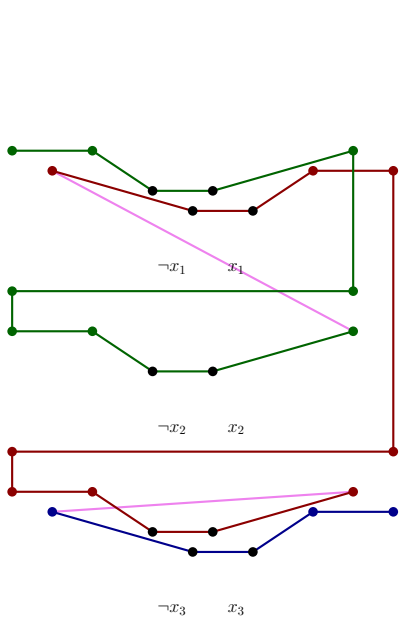
We can construct this gadget such that its vertices are on grid points of a polynomial-size grid with square length ϵ. The vertices b_1 and b_2 are already on the grid. Consider the grid points l_1, l_2 to the left of them with distance ϵ. Our shortcuts $s \to b_2$ and $b_1 \to t$ will lie on the lines defined by $\overline{l_1 b_2}$ and $\overline{b_1 l_2}$, respectively. Lengthen the line segments $\overline{l_1 b_2}$ and $\overline{b_1 l_2}$ at l_1 and l_2 by a factor of 2 (we use 1.5 in Figures 5 and 6 to keep it overseeable)—the endpoints are the grid points that will be our s and t. Observe that the shortcut $s \to t$ is always invalid if we place all variable-synchronization-gadgets in a column above the other with sufficient vertical spacing (constant in ϵ).

Complete Reduction

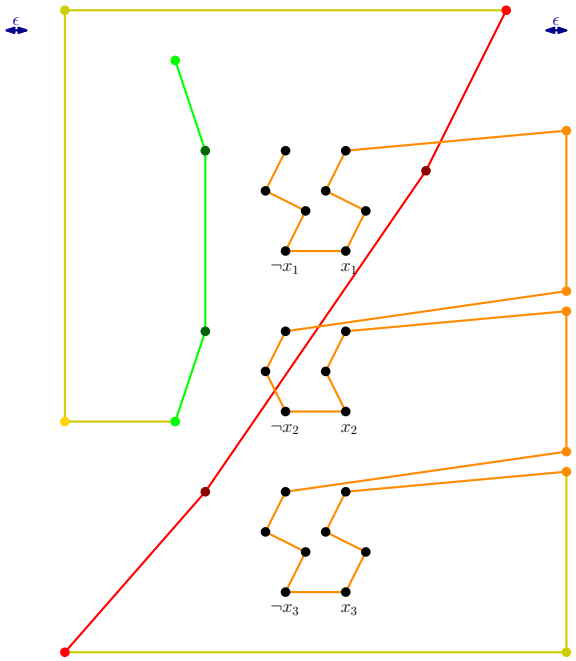
Given a Max-2-SAT instance, we can reduce it in polynomial time to a Polyline-Bundle-Simplification instance with goal Min-Edges: set ϵ to 1 and create for each variable a variable-synchronization-gadget with size equal to the number of occurrences of this variable and place them one above the other onto an integer grid. This defines exact positions for the literal-gadgets and then for the clause-synchronization-gadgets. Connect all literal-gadgets (first polyline) and, separately, all variable-synchronization-gadgets and clause-synchronization-gadgets (second polyline) in a shortcut-free way. This is possible on a polynomial-size grid. From a solution minimizing the number of edges of this polyline bundle simplification instance with two polylines, we can immediately obtain a solution of the corresponding Max-2-SAT instance—the total number of removed b-vertices equals the maximum number of satisfiable clauses. Thus, we conclude the following theorem:

▶ **Theorem 3.1.** Polyline-Bundle-Simplification *with goal* Min-Edges *is NP-hard even for two polylines.* ◀

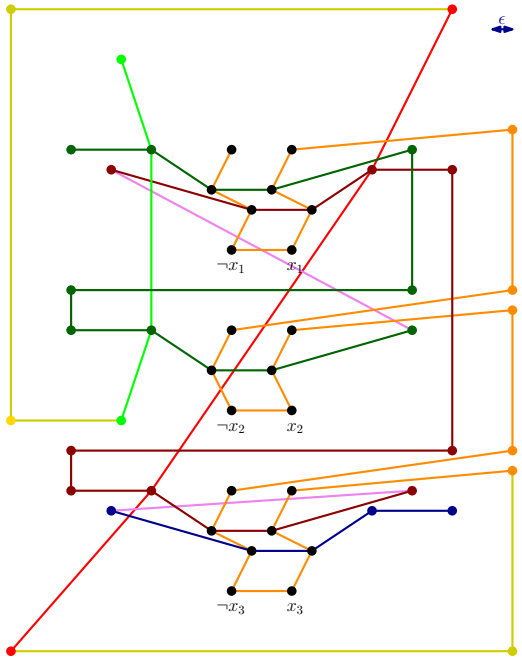
We give a small but full example to the presented reduction in Figure 6. We use the 2-SAT formula $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3)$. Note that the last clause $(\neg x_3)$ consists of only one literal and, therefore, does not have a clause-synchronization-gadget.



(a) first polyline (connecting literal-gadgets)



(b) second polyline (connecting both types of synchronization-gadgets)



(c) both polylines

Figure 6 Full example of our *NP*-hardness reduction: $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3)$

4 Fixed-Parameter Tractability

For both goals (Min-Edges or Min-Vertices) our problem is fixed-parameter tractable in the number of shared vertices, that is, vertices contained in more than one polyline or multiple times in the same polyline. We call the set of those vertices V_{shared} and let $k := |V_{shared}|$.

▶ **Theorem 4.1.** Polyline-Bundle-Simplification *is fixed-parameter tractable in* k.

Proof sketch. We sketch an algorithm that solves Polyline-Bundle-Simplification in $O(2^k \cdot \ell n^3)$ time. The idea is to fix for each subset $V' \subseteq V_{shared}$ the vertices in V' to be contained in V^* and the vertices in $V_{shared} \setminus V'$ to be excluded from V^*. Then the optimal simplification of the remaining parts, which are simple polylines, can be computed in the classic way [6]. In the end, we take the best solution among all 2^k subsets of V_{shared}. ◀

5 Conclusion

We have generalized the well-known problem of polyline simplification from a single polyline to multiple interfering polylines. Unlike the special case of a single polyline, simplifying a bundle of polylines turned out to be *NP*-hard. The problem is fixed-parameter tractable in the number of shared vertices, but not in the number of polylines.

The *NP*-hardness result gives rise to the question of approximability. It can be shown that the reduction from Max-2-SAT gives even *APX*-hardness. Therefore, it is an interesting question if there is a constant-factor approximation algorithm for our problem.

References

1 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5:75–91, 1995. URL: https://doi.org/10.1142/S0218195995000064, doi:10.1142/S0218195995000064.

2 W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry and Applications*, 6(1):59–77, 1996. URL: https://doi.org/10.1142/S0218195996000058, doi:10.1142/S0218195996000058.

3 David H. Douglas and Thomas 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.

4 Felix Hausdorff. *Grundzüge der Mengenlehre*. Veit and Company, Leipzig, 1914. URL: https://archive.org/details/grundzgedermen00hausuoft.

5 John Hershberger and Jack Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. In *Proc. 5th Intl. Symp. Spatial Data Handling (SDH'92)*, pages 134–143. IGU Commission on GIS, 1992.

6 Hiroshi Imai and Masao Iri. Polygonal approximations of a curve – formulations and algorithms. In Godfried T. Toussaint, editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 71 – 86. North-Holland, 1988. URL: http://www.sciencedirect.com/science/article/pii/B9780444704672500114, doi:https://doi.org/10.1016/B978-0-444-70467-2.50011-4.

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