SIMPLIFICATION OF POLYLINE BUNDLES OF GRAPHS AND TREES*

Yannick Bosch,[†] *Peter Schäfer,*[†] *Joachim Spoerhase,*[‡] *Sabine Storandt,*[†] *and Johannes Zink*[§]

Abstract. Polyline simplification is a well-studied optimization problem, in which a given polyline shall be replaced by a polyline with fewer vertices which still represents the shape of the original polyline faithfully. In this paper, we propose and study a generalization of the polyline simplification problem. Instead of a single polyline, we are given a set of ℓ polylines possibly sharing some line segments and vertices. We call such a set a *polyline bundle*. The task is to *simplify* each polyline L of a given polyline bundle by keeping a subset of its vertices such that (i) the Hausdorff or Fréchet distance between L and its simplified counterpart does not exceed a given distance threshold δ, (ii) a shared vertex is either kept or discarded in all polylines of the polyline bundle (we refer to this requirement as *consistency*) and (iii) the number of kept vertices in the polyline bundle is minimized. To justify this definition, we argue that consistency is crucial to get meaningful and aesthetically pleasing outputs.

Regarding the computational complexity of polyline bundle simplification, we prove that this problem is NP-hard to approximate within a factor of $n^{1/3-\varepsilon}$ for any $\varepsilon > 0$, where n is the number of vertices in the polyline bundle. This inapproximability even applies to planar inputs and also to instances with only $\ell = 2$ polylines. However, we identify the sensitivity of the solution to the choice of the distance threshold δ as a reason for this strong inapproximability. In particular, we prove that if we employ the Fréchet distance and allow δ to be exceeded by a factor of 2 in the solution, then we can find a simplified polyline bundle with no more than $\mathcal{O}(\log(\ell + n)) \cdot \mathsf{OPT}$ vertices in polytime, providing us with an efficient bi-criteria approximation. In addition, we show that the polyline simplification problem is solvable in polytime in case the polylines form a rooted tree. We further present a greedy heuristic that decomposes general bundles into tree bundles, which then can be simplified individually and optimally. In our experimental study, we compare the performance of the bi-criteria approximation algorithm and the tree bundle decomposition algorithm on public transit networks and movement trajectories. We show that in case the polylines form grid-like structures, the bi-criteria approximation algorithm outputs smaller simplifications, but the tree bundle decomposition algorithm scales better and produces superior results on polyline bundles derived from paths in embedded road networks.

*Preliminary versions of this work have appeared in
Proceedings of 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020) and in *Proceedings of 27th International Computing and Combinatorics Conference* (COCOON 2021).

[†] *University of Konstanz, Germany*
[‡] *University of Liverpool, United Kingdom*
[§] *University of Würzburg, Germany; Technical University of Munich, Heilbronn, Germany*

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 often means that the full data 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), which is a continuous sequence of line segments whose endpoints are called *vertices*. For such a polyline, a simplification is supposed to be as sparse as possible and as close to the original as necessary. Polyline simplification has many further applications, including data compression, trajectory clustering and analysis, as well as noise reduction in GPS-based movement sequences [37, 30, 5].

1.1 Simplifying a Single Polyline

A simplified polyline is usually constructed by a subset of vertices of the original polyline such that the distance to the original polyline does not exceed a specifiable value according to a given distance measure, e.g., the Fréchet distance or the Hausdorff distance. The first such algorithm, which is still of high practical importance, was proposed by Ramer [33] and by Douglas and Peucker [16]. Hershberger and Snoeyink [26] proposed an implementation of this algorithm that runs in $\mathcal{O}(n \log n)$ 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 in 1988 by Imai and Iri [27] for the Hausdorff distance. Its cubic runtime was shortly after improved to $\mathcal{O}(n^2 \log n)$ time by Melkman and O'Rourke [29], before in 1996, it was improved another time by Chan and Chin [13] to $\mathcal{O}(n^2)$ time. For the Fréchet distance, the optimal solution can be determined in $\mathcal{O}(n^3)$ time as described by Godau [22]. Recently, this runtime has been improved to $\mathcal{O}(n^2 \log n)$ [34] exploiting techniques introduced by Melkman and O'Rourke [29] and Guibas et al. [23].

We remark that all of these algorithms consider the distance *locally* (or *segment-wise*). This is, the distance between each line segment of the simplification and its corresponding subpolyline of the input polyline does not exceed the given threshold. We adhere to this widespread approach. Intuitively and from an application point of view, it makes sense to map a point p of the input polyline only to a point of a segment of the simplification "spanning over" p with respect to the input polyline as this ensures some degree of locality.

However, the general unrestricted approach has also received attention in the literature. Here, the Hausdorff or Fréchet distance between the input polyline and the simplification as a whole polyline is considered. For the (undirected) Hausdorff distance, this problem becomes NP-hard [36] and for the Fréchet distance, there is an $\mathcal{O}(n^3)$ time algorithm [6]. The problem variant where in addition the requirement is dropped that all vertices of the simplification must be vertices of the input polyline, is called a *weak* simplification. Agarwal et al. [1] showed that an optimal simplification under the local Fréchet distance with distance threshold δ, as computable using the algorithm by Imai and Iri, has no more vertices than an optimal weak simplification with distance threshold $\delta/4$; see also Section 1.3. We

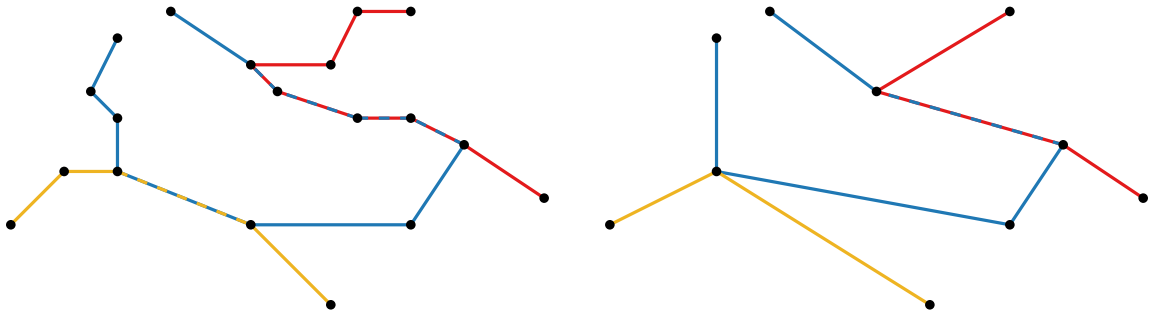


Figure 1: Example of a polyline bundle with three polylines (indicated by different colors) before and after consistent simplification.

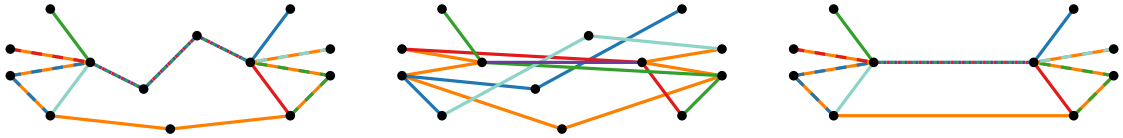


Figure 2: A polyline bundle (left) with six polylines, which are indicated by color, that are simplified independently (middle) and consistently (right). In the independently simplified polyline bundle, the individual polylines are inconsistent in which vertices they keep. This gives an unclearer picture with increased complexity (15 vs. 17 vs. 12 segments).

note that computing the Fréchet distance between two polylines can be solved in polynomial time [3], but may become NP-hard when considering additional properties like allowing to take shortcuts, which replace outliers in one of the polylines [10].

1.2 From a Single Polyline to a Polyline Bundle

On a map, there are usually multiple polylines to display. Such polylines may share vertices and line segments between vertices sectionwise. We call them a *polyline bundle*. For an illustration, see Figure 1. A good example is a schematic map of a public transport network where bus and metro lines are the polylines and these share some of the stations and legs. Other examples are trajectories of cars that are on the same roads for a while and then their paths may split and re-join, or the visualization of a flow network, where elementary flows may share edges and may separate or merge at vertices.

One might consider simplifying the polylines of a bundle independently. This has some drawbacks, though. On the one hand, the total complexity tends to increase when the shared parts are simplified in different ways; see Figure 2. On the other hand, it might suggest a misleading picture when we remove common segments 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. Thus, we require that a vertex in a simplification of a polyline bundle is either kept in all polylines containing it or discarded in all polylines. We call such a simplification *consistent*. Note that in Figure 1, the given polyline bundle is

simplified consistently. The objective is to minimize the total number of retained vertices.

1.3 Related Work

Agarwal et al. [1] describe an $\mathcal{O}(n \log n)$ time approximation algorithm for (classical) polyline simplification under the Fréchet distance. It is an approximation algorithm in the sense that the output simplification for distance threshold δ has at most as many vertices as an optimal solution with distance threshold $\delta/2$. We later also relate the size of our approximate solution respecting a distance threshold of δ for a polyline bundle to an optimal solution with distance threshold $\delta/2$.

A *bi-criteria approximation* is a generalization of a (classical) approximation where it is allowed to violate a certain constraint by a given factor. In particular, an algorithm is called a *bi-criteria* (α, β)*-approximation algorithm* if it runs in polynomial time and produces a solution of value at most $\alpha \cdot \mathsf{OPT}$, where OPT is the value of an optimal solution, while relaxing a constraint of the problem by a factor of β. There are two bi-criteria (α, β)-approximation algorithms for a weak simplification under the global Fréchet distance known, where α is the approximation factor for the number of retained vertices and β is the approximation factor for an allowed distance threshold violation: a (1, 8)-approximation in $\mathcal{O}(n \log n)$ time [1] and, for any $\varepsilon > 0$, a (2, 1+ε)-approximation in $\mathcal{O}(n^2 \log n \log\log n)$ time [35].

There is a multitude of polyline simplification problem variants for single polylines which involve additional constraints. One important variant is the computation of the smallest possible simplification of a single polyline which avoids self-intersection [15]. Another practically relevant variant is the consideration of topological constraints. For example, if the polyline represents a country border, important cities within the country should remain on the same side of the polyline after simplification. It was proven that those problem variants are hard to approximate within a factor of $n^{1/5-\varepsilon}$ [18]. Hence, in practice, they are typically tackled with heuristic approaches [18, 21].

Note that the only allowed inputs to those problem variants are either a single polyline without self-intersections or a set of polylines without self-intersections and without common vertices or segments (except for common start and end points). In contrast, we explicitly allow non-planar inputs and polyline bundles in which vertices and segments may be shared among multiple polylines. We also remark that the known results on hardness of approximation of these problems heavily rely on the constraint that feasible solutions are still non-intersecting. Since we do not require this, we have to resort to different techniques.

The so called *chain pair simplification problem* asks for the simplifications of two given polylines such that, for a given $k \in \mathbb{N}$ and $\delta > 0$, each simplified chain contains at most k segments, and the Fréchet distance between them is at most δ [4]. The problem arises in protein structure alignment or map matching tasks and was studied from a theoretical and practical perspective [38, 19, 20]. While the basic idea to preserve resemblance between polylines after simplification is similar to the motivation behind our problem of polyline bundle simplification, chain pair simplification only ever considers two polylines and does not put further restrictions on the simplification of shared parts.

The (k, λ)-*center (-median) clustering problem for polylines*[1] has been introduced by Driemel, Krivošija, and Sohler in 2016 [17]. Given a set $\mathcal{L}$ of polylines ($|\mathcal{L}| > k$), the problem asks for a set $\mathcal{C}$ of k polylines with λ vertices each, such that the maximum (the sum) of the Fréchet distances between each polyline $L \in \mathcal{L}$ and the polyline in $\mathcal{C}$ being closest to L is minimized. For polylines in two dimensions, there exists a 3-approximation for the (k, λ)-center clustering problem, but, even if $k = 1$, it is NP-hard to approximate within any factor smaller than 2.25 and $W[1]$-hard in the number of polylines [7, 8]. For polylines in d dimensions, there exists a randomized bi-criteria approximation algorithm for the (k, λ)-median clustering problem [9]. It approximates the solution in both, the Fréchet distance $(1 + \varepsilon)$ and, for the polylines in $\mathcal{C}$, the number of vertices $(2\lambda - 2)$. The running time of the algorithm is exponential in d, λ, $1/\varepsilon$, and a parameter for the failure probability. Cheng and Huang [14] improve this result by not increasing the number of vertices (i.e., theirs is not a *bi-criteria* approximation algorithm). Also, they provide an approximation algorithm for the following generalized polyline simplification problem. Given a number $\lambda \in \mathbb{N}$ and a set $\mathcal{L}$ of d-dimensional polylines with an individual distance threshold δ_i per polyline $L_i \in \mathcal{L}$, the task is to find a simplified polyline S with λ vertices such that, for each $L_i \in \mathcal{L}$, the Fréchet distance between L_i and S is at most δ_i. As a corollary, they obtain a bi-criteria approximation algorithm that approximates, for a single d-dimensional polyline, an optimal solution in both, the Fréchet distance $(1+\varepsilon)$ and the number of vertices $(1+\alpha)$ for fixed $\varepsilon, \alpha > 0$. Although these papers consider multiple polylines, there are quite some differences to polyline bundles. Most notably, their polylines do not explicitly share vertices and segments. Also, in the simplification step, we do not aim for fewer polylines and, consequently, a weak simplification. Instead, for every original polyline, there is a specific simplified polyline whose vertices are a subset of the original polyline.

Analyzing bundles of (potentially overlapping and intersecting) movement trajectories is an important means to study group behavior and to generate maps. For example, the RoadRunner approach [25] infers high-precision maps from GPS trajectories. Buchin, Kilgus, and Kölzsch [11] proposed an approach that computes a concise graph that represents all trajectories in a given set sufficiently well. However, these and similar methods do not produce valid simplifications of each input polyline, but allow to discard outliers or to let a polyline be represented by a completely disjoint polyline, which is quite different from our setting of polyline bundles. For more related work on map construction, which also uses the Fréchet distance, see the book by Ahmed, Karagiorgou, Pfoser, and Wenk [2].

1.4 Contribution

After giving some basic definitions on polylines and distance measures in Section 2, we introduce the problem of polyline bundle simplification in Section 2.4. Roughly speaking, we are given ℓ polylines on an underlying set P of n points that represent the vertices as well as an error bound δ and we seek to find a subset $P^\star$ of P such that, for each polyline L, the local Hausdorff or Fréchet distance between L and the simplified version of L where the vertices in $P \setminus P^\star$ have been removed is at most δ, and $|P^\star|$ is minimized.

[1] In literature, the term (k, ℓ)-*center (-median) clustering* is used. As we use ℓ already for the number of polylines in a polyline bundle, we use λ here instead to avoid confusion.

While the optimal simplification of a single polyline can be computed in polynomial time, we show in Section 3 that polyline bundle simplification is NP-hard to approximate within a factor of $n^{1/3-\varepsilon}$ for any $\varepsilon > 0$. This result applies already to bundles of two polylines, hence excluding an FPT-algorithm depending on parameter ℓ. We extend this hardness of approximation bound also to the case of *planar instances* where the polyline segments can only intersect in their endpoints.

On the positive side, we show in Section 4 that this strong inapproximability can be overcome when relaxing the error bound δ slightly. In particular, we design an efficient bi-criteria approximation algorithm. Here, we allow the simplified polylines in our solution to have a Fréchet distance of 2δ instead of only δ to the original polylines. We can then approximate the optimal solution for the original choice of δ within a factor logarithmic in the input size. As the choice of δ for real-world problems often is made in a rather ad-hoc fashion and uncertainties with respect to the precision of the input polylines have to be factored in as well, we deem our bi-criteria approximation to be of high practical relevance.

We show in Section 5 that, while the number of polylines in the bundle is not suitable to obtain an FPT-algorithm, the problem of polyline bundle simplification is fixed-parameter tractable in the number of vertices that are shared among the polylines.

We then study in Section 6 the special case of polyline bundle simplification where the polylines form a rooted tree and show that the resulting optimization problem can be solved optimally in polynomial time. Similar to the Imai–Iri algorithm [27] for simplification of a single polyline, our algorithm precomputes the possible set of shortcuts for the given distance threshold and thereupon transforms the given geometric problem into a graph problem. But while in the Imai–Iri algorithm a simple search for the minimum link-path in the shortcut graph suffices, we need a more intricate dynamic programming approach (DP) to deal with the tree structure. Furthermore, we devise a greedy heuristic that decomposes a polyline bundle into tree bundles, which then can be simplified to optimally with our DP.

In the experimental evaluation in Section 7, we use the bi-criteria approximation algorithm as well as the tree decomposition based approach to simplify polyline bundles that model movement data or public transit maps.

2 Preliminaries

Here, we give the basic definitions and notations for polylines, polyline bundles, the problem of finding a simplifications and the employed distance measures. Note that we define polyline(s) (bundles) as objects embedded in the plane since we only consider the 2-dimensional setting. However, one may generalize these definitions to an arbitrary number of dimensions.

2.1 Polylines

A *polyline* is a series of line segments that are defined by a sequence of points $L = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$, which we call *vertices*. Sometime a polyline is also called *polygonal chain*, *polygonal path*, *piecewise linear curve* or *linestring*. By n, we denote the *length* of a polyline, i.e., the number of vertices. The continuous curve induced by these vertices

is denoted as $c_L\colon [1,n] \to \mathbb{R}^2$ with $c_L\colon x \mapsto (\lfloor x \rfloor + 1 - x)p_{\lfloor x \rfloor} + (x - \lfloor x \rfloor)p_{\lceil x \rceil}$. The polyline simplification problem is defined as follows.

Definition 1 (Polyline Simplification)**.** *Given a polyline $L = (p_1, p_2, \ldots, p_n)$ and a distance threshold parameter δ, the objective is to obtain a minimum size subsequence S of L such that*

- *$p_1, p_n \in S$, and*
- *$d_X(L, S) \leq \delta$, where d_X is a distance measure for determining the distance between two polylines.*

We refer to S as a *simplification* of the *original* or *input* polyline L. Note that all vertices of S are also vertices of L.

2.2 Distance measures

Next, we introduce two classical candidates for a distance measure d_X used for polyline simplification. These are the Hausdorff and the Fréchet distance in their local and their global variant.

Definition 2 (Global Hausdorff Distance)**.** *Given two polylines $L_1 = (p_1, p_2, \ldots, p_n)$ and $L_2 = (q_1, q_2, \ldots, q_m)$, the* global (undirected) Hausdorff distance $d_{\mathrm{H}}(L_1, L_2)$ *is defined as*

$$d_{\mathrm{H}}(L_1, L_2) := \max\left\{ \sup_{p \in c_{L_1}} \inf_{q \in c_{L_2}} d(p,q), \sup_{q \in c_{L_2}} \inf_{p \in c_{L_1}} d(p,q) \right\},$$

where sup *is the supremum,* inf *is the infimum and $d(p,q)$ is the Euclidean distance between the points p and q.*

An often raised criticism concerning the use of the Hausdorff distance is that it does not reflect the similarity of the courses of two polylines. In contrast, the Fréchet distance measures the maximum distance between two polylines while traversing them in parallel and is therefore often regarded as the better suited measurement for polyline similarity.

Definition 3 (Global Fréchet Distance)**.** *Given two polylines $L_1 = (p_1, p_2, \ldots, p_n)$ and $L_2 = (q_1, q_2, \ldots, q_m)$, the* global Fréchet distance $d_{\mathrm{F}}(L_1, L_2)$ *is defined as*

$$d_{\mathrm{F}}(L_1, L_2) := \inf_{\alpha,\beta} \max_{t \in [0,1]} d(c_{L_1}(\alpha(t)) - c_{L_2}(\beta(t))),$$

where $\alpha\colon [0,1] \to [1,n]$ and $\beta\colon [0,1] \to [1,m]$ are continuous and non-decreasing functions with $\alpha(0) = \beta(0) = 1$, $\alpha(1) = n$, $\beta(1) = m$.

Traditionally, in the context of polyline simplification, the *local* Hausdorff and *local* Fréchet distance is used, which only measures the Hausdorff or Fréchet distance between each line segment of the simplification and its corresponding subpolyline in the original polyline.

Definition 4 (Local Hausdorff and Fréchet Distance)**.** *Given a polyline* $L = (p_1, p_2, \ldots, p_n)$ *and a simplification* $S = (p_1 = p_{s_1}, p_{s_2}, \ldots, p_{s_{|S|}} = p_n)$ *of* L*, the* local (undirected) Hausdorff distance $d_{\mathrm{lH}}(S, L)$ *is defined as*

$$d_{\mathrm{lH}}(S, L) := \max_{i \in 1, \ldots |S|-1} d_{\mathrm{H}}((p_{s_i}, p_{s_{i+1}}), L[p_{s_i}, p_{s_{i+1}}]) \,,$$

and the local Fréchet distance $d_{\mathrm{lF}}(S, L)$ *is defined as*

$$d_{\mathrm{lF}}(S, L) := \max_{i \in 1, \ldots |S|-1} d_{\mathrm{F}}((p_{s_i}, p_{s_{i+1}}), L[p_{s_i}, p_{s_{i+1}}]) \,,$$

where $(p_{s_i}, p_{s_{i+1}})$ *is the polyline of length two (i.e., the line segment) from* p_{s_i} *to* $p_{s_{i+1}}$ *and* $L[p_{s_i}, p_{s_{i+1}}]$ *is the (sub)polyline obtained by taking the substring of* L *from* p_{s_i} *to* $p_{s_{i+1}}$*.*

Observe that the Hausdorff distance is a lower bound for the Fréchet distance. Later, when considering the Fréchet distance, we may say that the distance threshold δ is respected or exceeded already in the Hausdorff distance since not exceeding δ for the Hausdorff distance is a necessary condition to not exceed δ for the Fréchet distance.

When using a local distance measure, we can tell for each pair of vertices p_i, p_j (for $1 \leq i < j \leq n$) in the original polyline independently whether a simplification may contain the line segment (p_i, p_j) or not by only computing the distance between the line segment (p_i, p_j) and its corresponding subpolyline. When considering such a pair (p_i, p_j) as a line segment for a simplification, we call it a *shortcut* or *shortcut segment*. If the distance between a shortcut and its corresponding subpolyline does not exceed the distance threshold δ, we call it a *valid* shortcut. Note that trivially (p_i, p_{i+1}) is always a valid shortcut. If, in a simplification, (p_i, p_j) is a line segments, then the vertices $p_{i+1}, \ldots, p_{j-1}$ do not occur in this simplification. Hence, we say these vertices are *skipped* by the shortcut (p_i, p_j). Since we only consider local distance measures here, we sometimes drop the word *local*.

2.3 Polyline Simplification Algorithms

The framework of the algorithm by Imai and Iri [27] from 1988 is the base for many polyline simplification algorithms. As we also use this algorithm (or descendant algorithms) here, let us briefly sketch its principle.

Given a polyline L with n vertices $p_1, \ldots, p_n$ and a distance threshold parameter δ, the polyline simplification algorithm by Imai and Iri proceeds in two phases. In the first phase, the *shortcut graph* is constructed. This graph has a node for each vertex of L and it has an edge between two nodes if and only if there is a valid shortcut between the corresponding two vertices of L. For the Hausdorff and the Fréchet distance it can be checked in $\mathcal{O}(n)$ time [3, 22] whether the distance between a line segment and a polyline having $\mathcal{O}(n)$ vertices exceeds δ. Hence, the total running time of the first phase amounts to $\mathcal{O}(n^3)$. In the second phase, a shortest path from the first node p_1 to the last node p_n is computed in the shortcut graph, which can be accomplished in $\mathcal{O}(n^2)$ time.

There are several algorithms working in the framework introduced by Imai and Iri but improving the cubic runtime of the first phase. For the local Hausdorff distance,

Melkman and O'Rourke [29] where the first to improve the runtime to $\mathcal{O}(n^2 \log n)$. Building upon their work, Chan and Chin [13] showed how to reduce the runtime to $\mathcal{O}(n^2)$.

As in the algorithm by Imai and Iri, Chan and Chin use each vertex p_i for $i \in \{1, \ldots, n\}$ as a starting point to traverse the rest of the polyline vertex by vertex. To determine all valid shortcuts originating at p_i in linear time, they maintain a cone-shaped region called *wedge* which is the area in which all valid shortcuts are required to lie. More precisely, the wedge is an angular region having its origin at p_i and being the intersection of all angular regions that define the areas where valid shortcuts may lie for each intermediate vertex. When traversing the polyline, the wedge iteratively becomes narrower. Since containment in the wedge can be checked in constant time and the wedge can be updated in constant time, the running time of this phase is $\mathcal{O}(n)$ per starting vertex p_i and $\mathcal{O}(n^2)$ in total. Note that this procedure may produce false positives, i.e., some shortcuts are added to the shortcut graph G_1 though not being valid. This problem is solved by repeating this whole process in reverse direction of the polyline. This way a second (false positive) shortcut graph G_2 is produced and the (real) shortcut graph G is the intersection of both, i.e., it has an edge if and only if this edge appears in both G_1 and G_2. Still, the running time of $\mathcal{O}(n^2)$ for the first phase matches the running time of the second phase, resulting in a total running time of $\mathcal{O}(n^2)$.

For the local Fréchet distance, less is known. First, Godau [22] showed how to use the local Fréchet distance within the Imai–Iri algorithm in $\mathcal{O}(n^3)$ time. Buchin et al. [12] proposed a data structure to apply this algorithm in $\mathcal{O}(n^{2.5+\varepsilon})$ time and space for any $\varepsilon > 0$. Adjusting the techniques introduced by Melkman and O'Rourke [29] to work for the Fréchet distance, and pursuing a similar approach as Guibas et al. [23], we have improved the running time to $\mathcal{O}(n^2 \log n)$ and the space consumption to $\mathcal{O}(n)$ [34]. As in the algorithm by Chan and Chin, a wedge is used to determine the cone-shaped area in which all valid shortcuts starting at p_i are required to lie. Additionally, a *wave front* (called *frontier* by Melkman and O'Rourke), which is a sequence of circular arcs within the wedge, is maintained in amortized logarithmic time. It splits the wedge into two parts – all valid shortcuts need to end in the part not containing p_i. To determine all valid shortcuts starting at a vertex p_i, a single sweep (in one direction) suffices.

2.4 Polyline Bundles

A *polyline bundle* $\mathcal{L}$ is a set of polylines $L_1, \ldots, L_\ell$ for some $\ell \geq 2$ that may share common vertices and (where two polylines share two subsequent vertices) line segments.

An instance of the *polyline bundle simplification* problem (from now on abbreviated by PBS) is specified by a triple $(P, \mathcal{L}, \delta)$, where $P = \{p_1, \ldots, p_n\}$ is a set of n points (*vertices*) in $\mathbb{R}^2$, a polyline bundle $\mathcal{L}$ using only vertices from from P, and a distance threshold parameter δ, which specifies a threshold for some distance measure d_X between original and simplified polyline bundle. Each polyline $L_i \in \mathcal{L}$ ($i \in \{1, \ldots, \ell\}$) is *simple* in the sense that all vertices of L_i are distinct points of P.

Definition 5 (Polyline Bundle Simplification (PBS))**.** *Given a triple $(P, \mathcal{L}, \delta)$, the objective is to obtain a minimum-size subset $P^\star \subseteq P$ of points, such that for each polyline $L_i \in \mathcal{L}$ its*

induced simplification S_i (which is $L_i \cap P^\star$ while preserving the order of points)

- *contains the start and the end point of L_i, and*
- *$d_X(L_i, S_i) \leq \delta$, i.e., for each original polyline and its simplification, a distance measure d_X for polylines is at most δ.*

Note that taking a subset of the vertices and keeping precisely these vertices in all polylines directly yields a consistent simplification of the polyline bundle. Moreover, there always exists a solution to every PBS instance since setting $P^\star = P$ is always possible.

In general, the input and the output polylines may intersect themselves or other polylines. If the input polylines do not intersect themselves or other polyline (apart from common vertices in P), we say the instance is *planar* and we call the problem *planar PBS*. Note that the output polyline in the planar PBS may contain intersections. If we required also the output polyline to be intersection free, the problem would become NP-hard even for one polyline [18].

3 Hardness of Polyline Bundle Simplification

In this section, we describe a polynomial-time reduction from *minimum independent dominating set* (MIDS) to (planar) PBS to show NP-hardness and hardness of approximation. The reduction applies to both, the local Hausdorff and the local Fréchet distance used as a distance measure for PBS.

3.1 Minimum Independent Dominating Set (MIDS)

In the MIDS problem, we are given a graph $G = (V, E)$, where V is the node[2] set and E is the edge set of G. We define $\hat{n} = |V|$ and $\hat{m} = |E|$. The objective is to find a set $V^\star \subseteq V$ of minimum cardinality[3] that is a dominating set of G as well as an independent set in G. A dominating set contains for each node v, v itself or at least one of v's neighbors. An independent set contains for each edge at most one of its endpoints. Halldórsson [24] has shown that MIDS, which is also referred to as *minimum maximal independent set*, is NP-hard to approximate within a factor of $|V|^{1-\varepsilon}$ for any $\varepsilon > 0$. In his proof, he uses a reduction from SAT to MIDS: from a SAT formula Φ, he constructs a graph such that an algorithm approximating MIDS would decide if Φ is satisfiable.

We observe that this reduction preserves the inapproximability gap of $|V|^{1-\varepsilon}$ even if Φ is a 3-SAT formula. Moreover, we observe that the number of edges in the graph constructed in this reduction by a 3-SAT formula is linear in the number of nodes. Thus, we conclude the following corollary and assume henceforth that we reduce only from sparse graph instances of MIDS, in other words, $\hat{m} \leq c\hat{n}$ for some sufficiently large constant c.

Corollary 1. *MIDS on graphs of $\hat{n}$ nodes and $\mathcal{O}(\hat{n})$ edges, i.e., sparse graphs, is NP-hard to approximate within a factor of $\hat{n}^{1-\varepsilon}$ for any $\varepsilon > 0$.*

[2]For a graph, we use the term *nodes* instead of *vertices* to distinguish them from the vertices of a polyline.
[3]Below, we use V' to denote some, not necessarily minimum, independent dominating set in G.

3.2 Reduction from MIDS to PBS

Next, we describe how to construct in polynomial time, for a given graph $G = (V, E)$ (i.e., an instance of MIDS), a specific PBS instance $(P, \mathcal{L}, \delta)$ with n vertices. Afterwards, we show that, if we could find a simplified polyline bundle of $(P, \mathcal{L}, \delta)$ where the number of retained vertices is at most $n^{1/3-\varepsilon}$ times the number of retained vertices in an optimal simplification for some $\varepsilon > 0$, then we could approximate MIDS within a factor of $n^{1-\varepsilon}$. We analyze the construction with respect to the Hausdorff distance, but observe that the arguments apply to the Fréchet distance as well.

In our construction, every node, every edge, and every neighborhood gets a separate polyline.[4] Hence, we have three types of polylines (*gadgets*) with specific properties. Each of our gadgets looks like a lengthy zigzag piece where shortcuts exist that skip almost all[5] of the vertices of the gadget. Skipping almost all vertices of a gadget can be interpreted as follows.

- In a *node gadget*, it means that a node is **not** in the (independent dominating) set V',
- in an *edge gadget*, it means that the independent set property is observed, and
- in a *neighborhood gadget*, it means that the dominating set property is observed.

The node gadgets extend vertically and they are arranged next to each other in some arbitrary order from left to right. The edge and neighborhood gadgets extend horizontally and they share a vertex with each node gadgets they correspond to.[6] Our construction is illustrated in Figure 3. Figures 3a, 3c and 3d show the individual gadgets and Figure 3e shows an example of how they look combined.

All gadgets are explained in detail next. We define our gadgets with respect to the distance threshold δ, which can be chosen arbitrarily. In our formulas, we also use some $\gamma \leq 2\delta/(30c\hat{n}^2 + 5)$, the constant $c = \hat{m}/\hat{n}$ (and w.l.o.g. $c \geq 1$), and for the horizontal distance of our node gadgets, we use some $x_{\text{spacing}} \geq (6c\hat{n}^2 + 2)3\delta$. When we speak of *shortcuts in a gadget*, we mean the (valid) shortcuts that would exist if we consider this gadget as a polyline on its own, simplified with distance threshold δ.

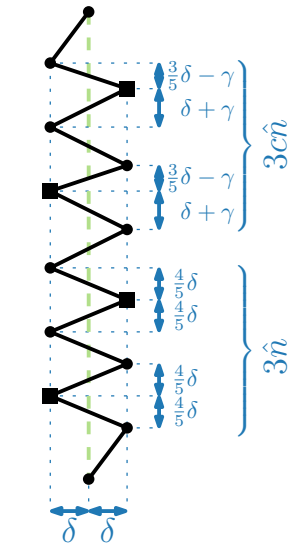
Note that our problem definition allows intersections and overlaps of different polylines without having a common vertex or segment (*non-planar input*). In this reduction, there can be such intersections, which, however, do not affect the involved polylines locally. In Section 3.3, we describe how to get rid of these intersections (*planar input*).

Node Gadget. For each node, we construct a *node gadget* (see Figure 3a). We arrange all node gadgets next to each other on a horizontal line in arbitrary order and with distance x_{spacing} between one and the next node gadget.

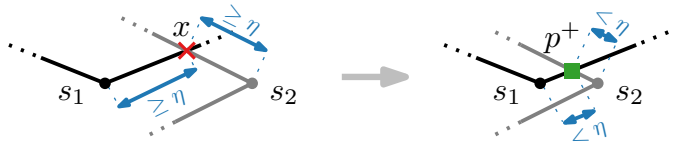
[4] These polylines can then be connected to have only two polylines in the bundle; see Section 3.6.

[5] We describe below what *almost all* means. Essentially, there is a large gap between skipping only a few and almost all vertices of a gadget.

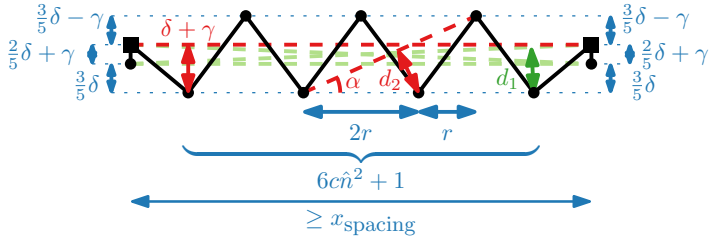
[6] An edge naturally corresponds to two nodes and a neighborhood corresponds to a set of nodes consisting of a node and all of its neighboring nodes.



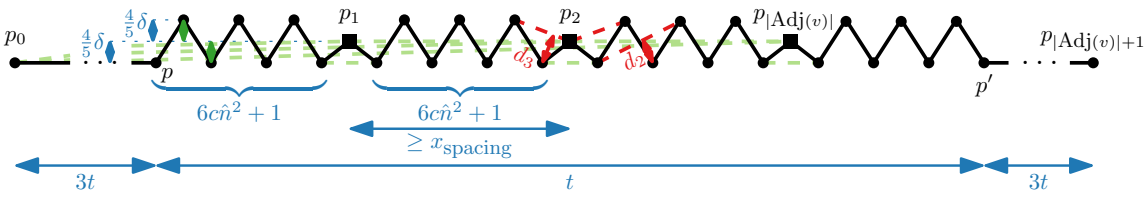
(a) Node gadget.



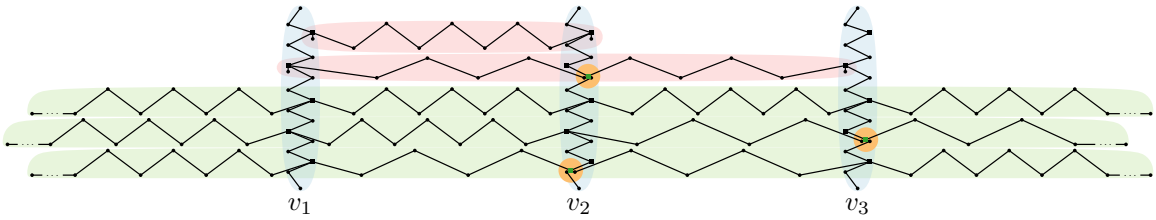
(b) Planarizing a polyline intersection: for an intersection point x, consider two adjacent vertices s_1 (on an edge or neighborhood gadget (black)) and s_2 (on a node gadget (gray)) with the same y-coordinate. We move s_1 towards s_2 until the distance between s_1 (s_2, respectively) and the intersection point is less than η. We then insert a new vertex p^+ (green square) onto this intersection point.



(c) Edge gadget.



(d) Neighborhood gadget.



(e) Combination of three node gadgets (for the nodes v_1, v_2, v_3; blue background) with two edge gadgets (for the edges v_1v_2 and v_1v_3; red background) and three neighborhood gadgets (for the nodes v_1, v_2, v_3; green background). Planarized crossings are highlighted by orange circles.

Figure 3: Schematization of our reduction from MIDS to planar PBS. Nodes shared by two gadgets are drawn as squares. Shortcuts are indicated by dashed green line segments. Dashed red line segments between two vertices indicate that there is no shortcut. The nodes in our minimum independent dominating set are precisely the ones for which we do not take the shortcut of the corresponding node gadgets.

A node gadget has $3\hat{n}(c+1)+2$ vertices arranged in a vertically-stretching zigzag course with x-distance 2δ (δ for the first and the last segment) between each two consecutive vertices. Each third vertex is a shared vertex, so there are $\hat{n}(c+1)$ shared vertices. The y-distance between a non-shared vertex and a shared vertex is $3/5\delta-\gamma$ and $\delta+\gamma$ in the upper part (i.e., the $3c\hat{n}+1$ topmost vertices of the gadget), and $4/5\delta$ in the lower part (i.e., the $3\hat{n}+1$ bottommost vertices of the gadget). This y-distance depends on whether the vertex is shared with an edge gadget (upper part) or a neighborhood gadget (lower part). We set the y-distance between each two neighboring non-shared vertices to 3δ.

Claim 1. *In a node gadget, there is precisely one shortcut, which starts at the first and ends at the last vertex.*

Proof. Clearly, the line segment s from the first to the last vertex has distance at most δ to the other vertices and segments of the node gadget, so this shortcut is valid.

It remains to show that there is no other shortcut. Any other potential shortcut segment would cross s at most once. Let this crossing be at a point o. We can assign vertices to the left of s only to the part of the shortcut segment above o and points to the right side of s only to the part of the shortcut segment below o – or the other way around. In both cases, when we traverse the zigzag piece, say, bottom-up, we encounter the vertices alternately on the left and on the right side of s, while the y-coordinates of consecutive vertices are strictly increasing. Thus, there cannot be another shortcut. ∎

We say that a node v is in V' if and only if we do not skip the inner vertices of the node gadget of v.

Edge Gadget. For each edge $\{u,v\}$, we construct an *edge gadget* (see Figure 3c) following a horizontally-stretching zigzag piece with $6c\hat{n}^2+5$ vertices and sharing its second and second last vertex with one of the two corresponding node gadgets – the node gadgets of u and v. All neighboring vertices from the second to the second last are equidistant in x-dimension, while the first and second vertex, and the second last and last vertex have the same x-coordinate. In y-dimension, the first and the last vertex are $2/5\delta+\gamma$ below the second and second last vertex, respectively. The other vertices are $3/5\delta-\gamma$ above the second vertex or $3/5\delta$ below the first vertex. In the following claim, we show that, in every edge gadget, there are three *long shortcuts*.

Claim 2. *In an edge gadget, there are precisely three long shortcuts. These are going (i) from the first to the last vertex, (ii) from the first to the second last vertex, and (iii) from the second to the last vertex. (iv) Beside these three (long) shortcuts, there are at most four more (short) shortcuts, which skip only the second and the second last vertex (and possibly also the third and third last vertex). (v) There is no shortcut not skipping one of the shared vertices, i.e., the second or the second last vertex.*

Proof. (i) The line segment from the first and to the last vertex is horizontal and has y-distance $3/5\delta$ or $2/5\delta+\gamma$ or δ to all inner vertices. (ii) Regarding a shortcut segment

from the first to the second last vertex, it is easy to see that the most critical part is the distance d_1 to the third last vertex. It is the y-distance between the third and the second last vertex, which is $3/5\delta + (2/5\delta + \gamma)$, minus at least a $(6c\hat{n}^2 + 2)$-th of the y-distance between the first and second last vertex, which is $2/5\delta + \gamma$. Combining these values yields

$$d_1 \leq \frac{3}{5}\delta + \left(\frac{2}{5}\delta + \gamma\right) - \frac{\frac{2}{5}\delta + \gamma}{6c\hat{n}^2 + 2} \leq \delta + \gamma - \frac{\frac{2}{5} \cdot \frac{30c\hat{n}^2+5}{2}\gamma + \gamma}{6c\hat{n}^2 + 2} = \delta\,.$$

Clearly, (iii) and (ii) are symmetric.

(v) If neither the second nor the second last vertex is skipped, then we cannot take any shortcut in this gadget. Clearly, we cannot take a "long" shortcut from the second to the second last vertex because the lower row of inner vertices has distance $\delta + \gamma$ from the potential shortcut segment.

Moreover, we cannot take a "short" shortcut from a vertex of the lower row to a non-neighboring vertex of the upper row or the other way around. Assume for a contradiction, we could skip two inner vertices. Then, the distance d_2 (see Figure 3c) from an inner vertex to the shortcut segment is at most δ. However, it is

$$d_2 = 2r \cdot \sin\alpha = 2r \cdot \sin\left(\arctan\frac{\frac{8}{5}\delta}{3r}\right) = 2r \cdot \frac{\frac{8\delta}{15r}}{\sqrt{\left(\frac{8\delta}{15r}\right)^2 + 1}} = \frac{16\delta r}{\sqrt{(8\delta)^2 + (15r)^2}}\,, \tag{1}$$

where r is the x-distance between two consecutive (inner) vertices. By construction,

$$r \geq \frac{x_{\text{spacing}}}{6c\hat{n}^2 + 2} \geq \frac{(6c\hat{n}^2 + 2)3\delta}{6c\hat{n}^2 + 2} = 3\delta\,, \tag{2}$$

and, hence,

$$d_2 \geq \frac{48\delta^2}{\sqrt{64\delta^2 + 2025\delta^2}} = \frac{48}{\sqrt{2089}}\delta = 1.0502\ldots\delta\,. \tag{3}$$

Observe that this becomes even greater if we want to skip four or more vertices instead of two vertices. Also it becomes greater if we start or end at one of the two shared vertices.

(iv) It remains to consider potential shortcuts starting or ending at the first or the last vertex. Clearly, skipping only the second or second last vertex is always possible. Skipping the second and the third vertex or skipping the second last and the third last vertex may sometimes be possible depending on how much the edge gadget is stretched horizontally. However, according to the previous analysis, skipping more vertices is not possible since the distance between the potential shortcut segment and the vertex before the endpoint of the potential shortcut is at least d_2. ∎

It follows that not skipping one of the two shared vertices is a relatively expensive choice in terms of retained vertices. Remember that not skipping one of the shared vertices means not taking the shortcut in the corresponding node gadget, which means putting the corresponding node into V'. So, skipping almost all vertices in the edge gadget of $\{u, v\}$ implies not having u or v in V', which means respecting the independent set property for the edge $\{u, v\}$.

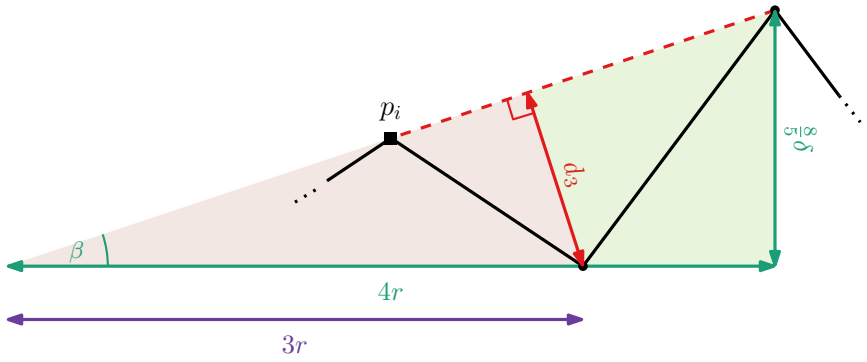


Figure 4: The potential shortcut segment in a neighborhood gadget form a vertex p_i to an inner vertex is dashed in red. However, due to $d_3 > \delta$, it is no valid shortcut segment.

Neighborhood Gadget. For each node v, we construct a *neighborhood gadget* (see Figure 3d). This gadget shares a vertex with every node gadget corresponding to a node of $\mathrm{Adj}(v)$, which is v and the nodes being adjacent to v. These shared vertices have the same y-coordinate.

The node gadgets of $\mathrm{Adj}(v)$ appear in an arbitrary horizontal order in our construction. Say the corresponding nodes in order are $u_1, \ldots, u_{|\mathrm{Adj}(v)|}$. Let the shared vertices with u_1 and $u_{|\mathrm{Adj}(v)|}$ be p_1 and $p_{|\mathrm{Adj}(v)|}$, respectively. We add a vertex p, which is placed x_{spacing} to the left and $4/5\delta$ below p_1. Symmetrically, we add a vertex p', which is placed x_{spacing} to the right and $4/5\delta$ below $p_{|\mathrm{Adj}(v)|}$. The vertices p and p' are the second and second last vertex of the neighborhood gadget. We place the first vertex of the neighborhood gadget, which we denote by p_0, on the same height and $3t$ to the left of p, where t is the distance between p and p'. Symmetrically, we place the last vertex of the gadget, which we denote by $p_{|\mathrm{Adj}(v)|+1}$, on the same height and $3t$ to the right of p'.

Between each two vertices p_i and p_{i+1} with $i \in \{0, \ldots, |\mathrm{Adj}(v)|\}$, we add a regular horizontally-stretching zigzag piece with $6c\hat{n}^2 + 1$ vertices (including p and p', excluding all p_i). The one half of the vertices of the zigzags is on the same height as p and p' and the other half is $8/5\delta$ above.

Claim 3. *In a neighborhood gadget, the only shortcuts (i) skip only p_i with $i \in \{1, \ldots, |\mathrm{Adj}(v)|\}$ or (ii) start at p_j with $j \in \{0, \ldots, |\mathrm{Adj}(v)|\}$ and end at p_k with $k \in \{j+1, \ldots, |\mathrm{Adj}(v)|+1\}$ except for the case that $j = 0$ while $k = |\mathrm{Adj}(v)| + 1$.*

Proof. (i) Clearly, for each $i \in \{1, \ldots, |\mathrm{Adj}(v)|\}$, the shortcut that starts at the vertex directly before p_i, skips only p_i, and ends at the vertex directly after p_i is valid.

(ii) For each $j \in \{1, \ldots, |\mathrm{Adj}(v)|-1\}$ and each $k \in \{j+1, \ldots, |\mathrm{Adj}(v)|\}$, there clearly is a valid shortcut from p_j to p_k. For $j = 0$ and each $k \in \{1, \ldots, |\mathrm{Adj}(v)|\}$, observe that, in the most extreme case, the line segment s from p_0 to $p_{|\mathrm{Adj}(v)|}$ has a y-distance to the upper row of vertices of

$$\frac{4}{5}\delta + \frac{t}{4t} \cdot \frac{4}{5}\delta = \delta$$

when s passes p in x-dimension. Thus, this shortcut is valid and this also holds for

$k < |\mathrm{Adj}(v)|$. For each $j \in \{1, \ldots, |\mathrm{Adj}(v)|\}$ and $k = |\mathrm{Adj}(v)| + 1$, this argument applies symmetrically. Obviously, there is no shortcut from p_0 to $p_{|\mathrm{Adj}(v)|+1}$ since the potential shortcut segment has distance $8/5\delta$ to the upper row of vertices.

It remains to argue that there are no more valid shortcuts. A shortcut starting and ending at a vertex on the upper or lower row is not possible because it would either be a horizontal segment, which has distance $8/5\delta$ to the other row, or the distance to some vertex in between would be at least d_2, which we have shown to be greater than δ in Equations (1) and (3). It is easy to see that there is no shortcut starting at p_0 and ending at some inner vertex of the upper or lower row. The same holds true for shortcuts starting at some inner vertex of the upper or lower row and ending at $p_{|\mathrm{Adj}(v)|+1}$.

Moreover, a shortcut segment starting (ending) at some p_i for $i \in \{1, \ldots, |\mathrm{Adj}(v)|\}$ and skipping one vertex would have a distance of d_3 to this vertex as depicted in Figure 4. Since d_3 is inside a rectangular triangle, we can determine d_3 by

$$d_3 = 3r \cdot \sin \beta \,,$$

where r is the x-distance between two consecutive (inner) vertices in the corresponding zigzag piece and β is an angle in another rectangular triangle and thus can be determined by

$$\beta = \arctan \frac{\frac{8}{5}\delta}{4r} = \arctan \frac{2}{5r'} \,,$$

where $r' = r/\delta$. Putting them together, we get

$$d_3 = 3r'\delta \cdot \sin\left(\arctan \frac{2}{5r'}\right) = 3r'\delta \cdot \frac{\frac{2}{5r'}}{\sqrt{1 - \frac{4}{25r'^2}}} = \frac{6}{\sqrt{25 - \frac{4}{r'^2}}}\delta \,.$$

Since $r' \geq 3$ (see Equation (2)), this means $d_3 \geq 1.2108\ldots\delta$.

If we skip more than one inner vertex, the distance to the last skipped vertex becomes even greater than d_3. Hence, we conclude that the claim is correct. ∎

Due to Claim 3, we can skip almost all vertices in a neighborhood gadget if we keep at least one vertex from $p_1, \ldots, p_{|\mathrm{Adj}(v)|}$, which are the vertices shared with the node gadgets. If we skip all of them, we can skip no other vertex. So, to avoid high costs in terms of retained vertices, we must not take the shortcut of the node gadget of at least one node in $\mathrm{Adj}(v)$. This means that we must, for each $v \in V$, add a node of $\mathrm{Adj}(v)$ to V', which enforces the dominating set property.

3.3 Making the PBS Instance Planar

The current construction is (to a high degree) non-planar. We next describe how to make it planar, i.e., polylines cross each other only in common vertices. The key idea is to planarize the non-planar construction by replacing polyline intersections by new vertices, which we call *crossing vertices*. However, we need to be careful where to insert crossing vertices. Just

inserting vertices wherever a intersection point occurs could give rise to new shortcuts and hence destroy the mechanics of the gadgets. We can prevent this from happening if we ensure that crossing vertices lie sufficiently close to existing vertices. Then, we cannot gain anything by ending a shortcut at a crossing vertex rather than an original vertex.

First we increase the horizontal distance between each two neighboring node gadgets by a factor of 2. Now, every zigzag piece in every edge gadget and every neighborhood gadget (see Figures 3c and 3d) has width at least $2x_{\text{spacing}}$ instead of "just" x_{spacing}. This stretches all edge and neighborhood gadgets horizontally and gives us enough flexibility to move each individual inner vertex of these zigzag pieces (up to $0.5r$) to the left or to the right, while maintaining the functionality of the gadgets. Observe that it is a crucial property of our hardness construction that stretching edge and neighborhood gadgets horizontally does not change the behavior in terms of possible shortcuts because we have already assumed that x_{spacing} is only a lower bound for the width; see Claims 2 and 3.

Now consider an intersection point x (in the stretched drawing) and its two closest non-shared vertices s_1 and s_2, where s_1 lies in a zigzag piece of an edge or a neighborhood gadget and s_2 lies in a node gadget; see Figure 3b on the left. First note that s_1 and s_2 share a common y-coordinate by construction (the y-coordinates of the vertices in the upper part of a node gadget coincide with the y-coordinates of the vertices in the edge gadgets, and in the lower part they coincide with the neighborhood gadgets). We move s_1 horizontally towards s_2 such that the distance of the (also moving) intersection point to both s_1 and s_2 is less than η, which we specify below. Onto this carefully arranged intersection point, we now insert a new crossing vertex p^+ to planarize the construction; see Figure 3b on the right. Now, p^+ is a vertex of both involved gadgets.

For the correctness of Claims 2 and 3, we require that the horizontal distance between each two vertices in a zigzag piece of an edge or neighborhood gadget is $\geq 3\delta$; see Equation (2). Since we have increased this horizontal distance to $\geq 6\delta$ by horizontal stretching with factor 2 and we have moved s_1 by at most $0.5 \cdot 6\delta$, the horizontal distance of s_1 to its neighbors within the zigzag piece is still $\geq 3\delta$.

We now analyze how close to s_1 and s_2, the crossing vertex p^+ needs to be placed. We require s_1 and s_2 to be strictly inside a disk of radius η around p^+ to prevent the emergence of new shortcuts. Intuitively, η is chosen sufficiently small to ensure that, given any pair of vertices (p, q) that do not admit a valid shortcut, moving p or q within a disk of radius η does not bring the line segment (p, q) into the radius-δ neighborhood disk of some third vertex o. More formally, we let

$$\eta = \left(\min_{\substack{\{p,o,q\} \subseteq L, \\ L \in \mathcal{L}}} \{ d((p,q),o) \mid d((p,q),o) > \delta \} \right) - \delta \,.$$

Observe that we can determine η in polynomial time.

By Lemma 1, we show that the new crossing vertices do not allow new shortcuts and, hence, the functionality of the gadgets is not affected regardless of whether we keep the crossing vertex and skip the neighboring original vertices, which we call its *skip vertices*, or the other way around. For a set of shortcuts Z, we let $\mathsf{P}(Z)$ denote the set of endpoints

of all shortcuts in Z.

Lemma 1. *Let p^+ be a crossing vertex, and let Z_{p^+}, Z_{s_1}, and Z_{s_2} be the set of shortcuts having p^+ and p^+'s two skip vertices s_1 and s_2 as an endpoint, respectively. Then, $\mathsf{P}(Z_{p^+}) \setminus \{p^+\} \subseteq \mathsf{P}(Z_{s_1}) \cup \mathsf{P}(Z_{s_2})$.*

Proof. We prove this statement by contradiction. Suppose there is a vertex q such that the line segment (p^+, q) is a shortcut, whereas (s_1, q) and (s_2, q) are no shortcuts. W.l.o.g., let p^+, q, s_1 be vertices of the same polyline L. We know that $d_{\mathrm{H}}((p^+, q), L[p^+, q]) \leq \delta$.

For all of the gadgets, it has been shown that, wherever there is no shortcut between two vertices p and q, this is because some vertex o between p and q has Euclidean distance greater than δ to (p, q); see Claims 1 to 3. Hence in our case and by the choice of η, there is a vertex o on $L[s_1, q]$ (and thus also on $L[p^+, q]$) with $d((s_1, q), o) \geq \delta + \eta$.

By the choice of p^+, we know that $d_{\mathrm{H}}((p^+, q), (s_1, q)) < \eta$. For any vertex o' on $L[p^+, q]$, this implies, by using the triangle inequality, $d((s_1, q), o') < \delta + \eta$. This is a contradiction to the choice of o. ∎

Also note that we can always skip a crossing vertex as it lies on the line segment between its predecessor vertex and successor vertex on both of its polylines. Hence, we do not count crossing vertices in Section 3.5.

3.4 Size of the PBS Instance

Observe that all shared vertices are shared between only two polylines – by a node gadget and either an edge gadget or a neighborhood gadget. A node gadget provides enough vertices that may be shared with the edge and neighborhood gadgets as a node is contained in at most $\hat{n}$ neighborhoods and there are at most $c\hat{n}$ edges. In the following lemma, we analyze the size of the constructed planar PBS instance.

Lemma 2. *By our reduction, we obtain from an instance $G = (V, E)$ of MIDS an instance of PBS with a planar polyline bundle that has $n < 50c^2\hat{n}^3$ vertices, where $\hat{n} = |V| \geq 2$, $|E| = c\hat{n}$ ($c \geq 1$ is constant).*

Proof. By construction, we have at most one shared vertex for each pair of node gadget and edge gadget, and for each pair of node gadget and neighborhood gadget. This is a shared vertex either because the corresponding node is incident to the corresponding edge or part of the corresponding neighborhood, or because it is a crossing vertex. So in total we have at most $\hat{n} \cdot (\hat{m} + \hat{n})$ shared vertices. All node gadgets together have $\hat{n}(3c\hat{n} + 3\hat{n} + 2)$ vertices.

For the edge and neighborhood gadgets, we count only non-shared vertices because in the following sum, we add the shared vertices separately. For the node gadgets, we have counted all vertices because not all of its (potentially shared) vertices need to be shared. All edge gadgets have $\hat{m}(6c\hat{n}^2 + 3)$ non-shared vertices, and all neighborhood gadgets have

$(2\hat{m}+2\hat{n})\cdot(6c\hat{n}^2+1)+2\hat{n}$ non-shared vertices. Summing these values up and using $\hat{m}=c\hat{n}$ yields

$$\begin{aligned} n &\leq \hat{n}\cdot(\hat{m}+\hat{n})+\hat{n}(3c\hat{n}+3\hat{n}+2)+\hat{m}(6c\hat{n}^2+3)+(2\hat{m}+2\hat{n})\cdot(6c\hat{n}^2+1)+2\hat{n} \\ &= (18c^2+12c)\hat{n}^3+(4c+4)\hat{n}^2+(5c+6)\hat{n} < 50c^2\hat{n}^3\,. \end{aligned}$$ ∎

3.5 Correctness

We say a simplification of an instance of PBS obtained by this reduction *corresponds* to an independent and dominating set V' and vice versa if we take the (unique) shortcuts in the node gadgets except for the ones corresponding to V' and we skip all inner non-shared vertices (the zigzag pieces) in all edge and neighborhood gadgets, which is possible since V' is independent and dominating. Observe that for each independent and dominating set, there is precisely one corresponding simplification (which is also valid according to δ).

Lemma 3. *Let V' be a solution for an instance $G=(V,E)$ of MIDS. In the instance $(P,\mathcal{L},\delta)$ of PBS obtained by our reduction, the size of the simplification corresponding to V' is $\hat{n}(3(c+1)|V'|+2c+4)$, where $\hat{n}=|V|$ and $c\geq 1$ is constant.*

Proof. For all vertices except for the ones in V', we take the shortcuts in the corresponding node gadgets in $(P,\mathcal{L},\delta)$, which reduces the number of vertices in each of these gadgets to 2. This gives us $(\hat{n}-|V'|)\cdot 2+|V'|\cdot(3\hat{n}(c+1)+2)=\hat{n}\,(2+3(c+1)|V'|)$ remaining vertices in all node gadgets combined. In the following, we count shared vertices for the node gadgets. We take a "long" shortcut in all of the edge gadgets. This gives us two remaining non-shared vertices in all edges gadgets ($2c\hat{n}$ vertices in total). Moreover, we skip all inner non-shared vertices in all of the neighborhood gadgets ($2\hat{n}$ vertices remaining). Altogether, this sums up to $\hat{n}(3(c+1)|V'|+2c+4)$. ∎

By Lemma 3, we know that for an optimal solution $V^\star$ of an instance of MIDS, the corresponding simplification in the instance $(P,\mathcal{L},\delta)$ of PBS obtained by our reduction has size $\hat{n}(3(c+1)\mathsf{OPT}_{\text{MIDS}}+2c+4)$, where $\mathsf{OPT}_{\text{MIDS}}=|V^\star|$ and which of course is at least the size $\mathsf{OPT}_{\text{PBS}}$ of the optimal solution of $(P,\mathcal{L},\delta)$. We formalize this in the following lemma.

Lemma 4. *For an instance $G=(V,E)$ of MIDS and the instance $(P,\mathcal{L},\delta)$ of PBS obtained by our reduction from G, $\mathsf{OPT}_{PBS}\leq\hat{n}(3(c+1)\mathsf{OPT}_{MIDS}+2c+4)$.*

Theorem 1. *PBS with a planar polyline bundle as input is NP-hard to approximate within a factor of $n^{1/3-\varepsilon}$ for any $\varepsilon>0$, where n is the number of vertices in the polyline bundle. This hardness applies to the local Hausdorff and the local Fréchet distance used as a distance measure for PBS.*

Proof. Suppose for a contradiction that there is an approximation algorithm $\mathcal{A}$ solving any instance of PBS within a factor of $n^{1/3-\varepsilon}$ for some constant $\varepsilon>0$ with respect to the optimal solution. We can transform any instance $G=(V,E)$ of MIDS, where $\hat{n}=|V|,\hat{m}=|E|$, and $\mathsf{OPT}_{\text{MIDS}}=|V^\star|$, to an instance $(P,\mathcal{L},\delta)$ of PBS using the reduction described above,

where $|P| = n$ and the size of an optimal solution is $\mathsf{OPT}_{\mathrm{PBS}}$. This reduction clearly applies to both, the local Hausdorff and the local Fréchet distance.

Employing $\mathcal{A}$ to solve $(P, \mathcal{L}, \delta)$ yields a (simplified) polyline bundle $\mathcal{S}_{\mathcal{A}}$. We denote the number of vertices in $\mathcal{S}_{\mathcal{A}}$ by $n_{\mathcal{A}}$ and we know that $n_{\mathcal{A}} \leq \mathsf{OPT}_{\mathrm{PBS}} \cdot n^{1/3-\varepsilon}$ for some $\varepsilon > 0$. Suppose in $\mathcal{S}_{\mathcal{A}}$, there was a $(6c\hat{n}^2 + 1)$-vertex zigzag piece of an edge or neighborhood gadget (see Figures 3c and 3d) that was not skipped. Then, because the two end vertices of every gadget can also not be skipped and there are $\hat{n} + c\hat{n} + \hat{n}$ gadget in total, it would be $n_{\mathcal{A}} \geq 6c\hat{n}^2 + 1 + 2(\hat{n} + c\hat{n} + \hat{n}) > 6c\hat{n}^2 + (2c + 4)\hat{n}$. However, since there exists some independent dominating set for G containing at most $\hat{n}$ nodes, there exists a simplification of size at most $\hat{n}(3(c + 1)\hat{n} + 2c + 4) \leq 6c\hat{n}^2 + (2c + 4)\hat{n}$ due to Lemma 3 (recall that $c \geq 1$). Hence, we can assume that all zigzag pieces of the edge and neighborhood gadgets are skipped in $\mathcal{S}_{\mathcal{A}}$ (otherwise we can replace $\mathcal{S}_{\mathcal{A}}$ by a solution with fewer vertices by finding any independent dominating set of G greedily in polynomial time), and therefore, we can immediately read an independent dominating node set $V' \subseteq V$ from the node gadgets where the shortcut is not taken.

Using our assumption together with Lemma 3 and Lemma 4, we can state that

$$n^{\frac{1}{3}-\varepsilon} \geq \frac{n_{\mathcal{A}}}{\mathsf{OPT}_{\mathrm{PBS}}} \geq \frac{\hat{n}(3(c+1)|V'| + 2c + 4)}{\hat{n}(3(c+1)\mathsf{OPT}_{\mathrm{MIDS}} + 2c + 4)} > \frac{|V'|}{\mathsf{OPT}_{\mathrm{MIDS}} + \frac{2c+4}{3(c+1)}},$$

which we can reformulate as $|V'| < n^{1/3-\varepsilon}(\mathsf{OPT}_{\mathrm{MIDS}} + \frac{2c+4}{3(c+1)})$. We assume that $\mathsf{OPT}_{\mathrm{MIDS}} > \frac{2c+4}{3(c+1)}$ as otherwise we could check all subsets of V of size at most $\frac{2c+4}{3(c+1)}$ in polynomial time. Similarly, we can assume that $\hat{n}$ is sufficiently large to satisfy $\hat{n}^{2\varepsilon} > 100c^2$. Beside this, we apply Lemma 2 and obtain

$$\begin{aligned} |V'| &< n^{\frac{1}{3}-\varepsilon} \cdot 2 \cdot \mathsf{OPT}_{\mathrm{MIDS}} < 2 \cdot (50c^2\hat{n}^3)^{\frac{1}{3}-\varepsilon} \cdot \mathsf{OPT}_{\mathrm{MIDS}} \\ &< 100c^2 \cdot \hat{n}^{1-3\varepsilon} \cdot \mathsf{OPT}_{\mathrm{MIDS}} < \hat{n}^{2\varepsilon} \cdot \hat{n}^{1-3\varepsilon} \cdot \mathsf{OPT}_{\mathrm{MIDS}} = \hat{n}^{1-\varepsilon} \cdot \mathsf{OPT}_{\mathrm{MIDS}}. \end{aligned}$$

Since we know that it is NP-hard to approximate MIDS within a factor of $\hat{n}^{1-\varepsilon}$ for any $\varepsilon > 0$, it follows that $\mathcal{A}$ cannot be a polynomial-time algorithm unless P = NP. Or in other words, it is NP-hard to approximate PBS within a factor of $n^{1/3-\varepsilon}$ for any $\varepsilon > 0$. ∎

3.6 Using only Two Polylines

Currently, we use one polyline per gadget; refer to Figure 3e for a scheme of the current construction. So, our reduction uses $(2+c)\hat{n}$ polylines in total. We can reduce the number of polylines to two by connecting all node gadgets from left to right in a row (alternating with the connecting segments between bottom and top side), which gives us the first polyline, and by connecting all edge and neighborhood gadgets similarly, which gives us the second polyline.

For the latter, directly adding connecting segments between these gadgets can be problematic because new shortcuts or crossings may be created when the horizontal span of two such gadgets is very different and when their end vertices lie between node gadgets.

The neighborhood gadgets are already relatively long and might reach to the left of the leftmost node gadget and to the right of the rightmost node gadget. If not, we can simply stretch the two outermost zigzag pieces horizontally without violating the functionality of the gadgets (see Section 3.3). Then, we can simply connect the endpoints of the neighborhood gadgets with additional segments without creating new shortcuts.

The edge gadgets, however, have their endpoints between the node gadgets. The solution is to extend them to reach to the left and the right of all node gadgets similar to the neighborhood gadgets. There, we can connect them without creating new crossings or shortcuts. As for the neighborhood gadgets, we can do this for the edge gadgets by adding two additional zigzag pieces – one before the first and one after the last vertex of the edge gadget, which cross all node gadgets to the left and to the right. This does not violate the functionality of the edge gadget (in particular, consider the case that the two shared vertices of an edge gadget are kept). Observe that this also does not affect the approximation ratio asymptotically. Overall, we conclude the following corollary.

Corollary 2. *PBS is not fixed-parameter tractable (FPT) in the number of polylines ℓ. In particular, PBS with two polylines is already NP-hard to approximate within a factor of $n^{1/3-\varepsilon}$ for any $\varepsilon > 0$. This holds true even for planar polyline bundles and for both the local Hausdorff and the local Fréchet distance.*

4 Bi-criteria Approximation for Polyline Bundles of General Graphs

In this section, we describe a bi-criteria approximation algorithm for PBS under the local Fréchet distance. More precisely, it is a bi-criteria (α, β)-approximation algorithm in which we exceed the number of retained vertices by a factor of at most $\alpha \cdot \mathsf{OPT}$ and relax the error bound δ by a factor of β.

In Section 3, we have shown that there is no bi-criteria $(n^{1/3-\varepsilon}, 1)$-approximation algorithm for PBS for any $\varepsilon > 0$ unless P = NP. This strong inapproximability comes from the high sensitivity towards choices of keeping or discarding single vertices, which is modulated by the given value of δ. By making a bad choice, we cannot take (arbitrarily long) shortcuts that have a distance just a little greater than the given distance threshold δ to the original subpolyline. This can be overcome by relaxing the distance constraint slightly. In particular, we show that allowing a constraint violation by a factor of $\beta = 2$, we can design an efficient algorithm with an approximation guarantee of $\alpha \in \mathcal{O}(\log(\ell + n))$. For an illustration of our algorithm see Figure 7.

The key building block of our algorithm is a connection between PBS and a certain geometric set cover problem, which we call *star cover problem*. The star cover problem models the aspect of shortcutting polylines by few vertices but does not take into account consistency among different polylines. We argue, however, that approximate solutions to the star cover problem can be post-processed to form consistent PBS solutions by slightly violating the error threshold δ.

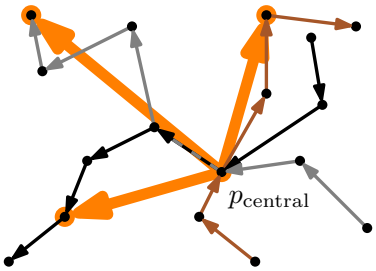


Figure 5: A *star* (in orange) around a vertex p_{central}, which lies on three polylines. Each polyline was assigned an arbitrary direction indicated by arrow heads.

4.1 Star Cover Problem (StCo)

Next, we introduce the *star cover problem*, which is a special type of the set cover problem defined over instances of PBS. Informally spoken, a star is a vertex together with incident (outgoing) shortcut segments of the polylines containing this vertex. To obtain a set of stars, we first direct each polyline $L \in \mathcal{L}$ in a given PBS instance $(P, \mathcal{L}, \delta)$ arbitrarily. We orient all shortcut segments of L in the same direction as L. Later we want to cover all segments of all polylines by shortcuts of stars. By directing the polylines, we can define for every vertex a unique "maximal" star and, moreover, when combining all of these stars, we can be sure that all segments are covered.

First, we define stars form formally; see Figure 5 for an example of a star.

Definition 6 (Star)**.** *A* star *is the combination of a vertex $p_{central} \in P$ and, for each polyline $L \in \mathcal{L}$ that contains $p_{central}$, one or zero outgoing shortcut segments with respect to the distance threshold δ.*

We say a star s *covers* a segment–polyline pair (e, L), if s contains a directed shortcut $(p_{\text{central}}, p_{\text{outer}})$ for L and e is a line segment of L coming somewhere between p_{central} and p_{outer} when traversing L. Our objective is to find a small set of stars that cover all segment–polyline pairs. We denote the set of all segment–polyline pairs in the input by $\mathcal{U}$ and the subset of segment–polyline pairs covered by a particular star s by $\mathcal{U}_s$. Then the star cover problem is defined as follows.

Definition 7 (Star Cover Problem (StCo))**.** *A* star cover *C is a set of stars such that $\bigcup_{s \in C} \mathcal{U}_s = \mathcal{U}$, i.e., all segment–polyline pairs are covered. The* star cover problem *asks for a minimum size star cover.*

Implicitly, a StCo instance depends on the distance measure we use for determining shortcuts. In this section, we only use the local Fréchet distance.

4.2 Relationship between Solutions of PBS and StCo

Next, we investigate the relationship between an instance of PBS and a corresponding instance of StCo. (Note that to one instance of PBS, there are different instances of

StCo differing only in the direction of polylines and shortcuts.) We argue that every (optimal) solution for PBS can be decomposed into a star cover. Hence, an optimal StCo solution yields a lower bound for an optimal PBS solution.

Lemma 5. *The size* OPT_{StCo} *of an optimal solution to an instance of StCo obtained from an instance* $(P, \mathcal{L}, \delta)$ *of PBS satisfies* $\mathsf{OPT}_{StCo} \leq \mathsf{OPT}_{PBS}$*, where* OPT_{PBS} *is the size of an optimal solution to* $(P, \mathcal{L}, \delta)$*.*

Proof. Consider an optimal solution $P^\star$ to $(P, \mathcal{L}, \delta)$. From the simplified polyline bundle induced by $P^\star$, we can get a star cover for any instance of StCo obtained from $(P, \mathcal{L}, \delta)$. First, orient all shortcuts of the simplified polyline bundle in the direction given by the StCo instance (between two vertices of $P^\star$, we may have shortcut segments in both directions on different polylines). Then, iteratively add a star in the following way. Pick a vertex $p_{\text{central}} \in P^\star$ that has at least one outgoing shortcut in the simplified polyline bundle – p_{central} becomes the central vertex of a star s. Attach to p_{central} all its outgoing shortcuts in the simplified polyline bundle to obtain the star s. Remove all shortcuts in s from the simplified polyline bundle. Repeat this procedure until there is no vertex with outgoing shortcut segments left in the simplified polyline bundle. Clearly, the obtained set of stars covers all segment-polyline pairs in $\mathcal{L}$ and is therefore a star cover. Clearly, it has at most $|P^\star|$ stars. Hence $\mathsf{OPT}_{\text{StCo}} \leq \mathsf{OPT}_{\text{PBS}}$. ∎

4.3 Approximation for StCo

We can compute an approximate solution for StCo by employing the classic greedy algorithm [28] for set cover, which iteratively selects the set with the most uncovered elements until all elements are covered. However, if applied naively for StCo, the running time would be exponential in the size of the PBS instance as the number of stars can be in the order of $\Omega(n^\ell)$ (as, in the worst case, a star has $\Theta(n)$ choices for the endpoint of a shortcut on each of $\Theta(\ell)$ polylines). Notice, however, that it suffices to consider only *maximal stars*. A *maximal star* s uses, for each polyline L containing the central vertex of s, the outgoing shortcut segment that covers the most segments of L. As there are only n maximal stars, this guarantees polynomial running time.

Lemma 6. *An* $\mathcal{O}(\log(t + w))$*-approximation for an instance of StCo obtained from an instance* $(P, \mathcal{L}, \delta)$ *of PBS can be computed in* $\mathcal{O}(\ell n^2 \log n)$ *time, where* t *is the maximum number of polylines any vertex occurs in and* w *is the maximum number of segments any valid shortcut (according to* δ*) can skip.*

Proof. There is a polynomial time greedy algorithm that yields an $\mathcal{O}(\log m)$ approximation for the set cover problem, where m is the size of the largest set in the given collection of subsets of the universe [28]. The greedy algorithm works as follows. While there are uncovered elements from the universe, add the set with the largest number of uncovered elements to the set cover. In an instance of StCo, m is the maximum number $\max_{\text{star } s} |\mathcal{U}_s|$ of segment–polyline pairs a single star can cover. If the central vertex of a star lies in at most t polylines, the star contains at most t shortcut segments, each of which covers at most

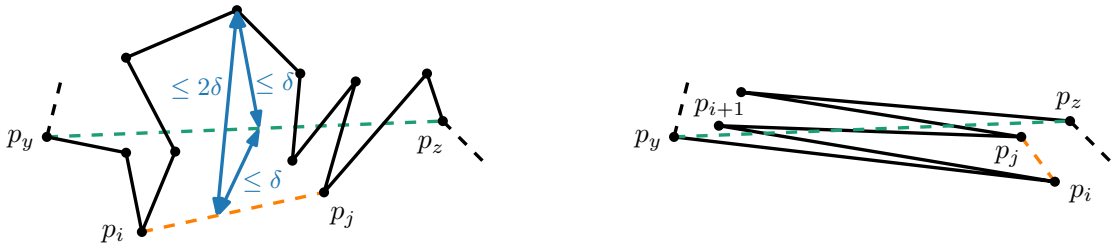


Figure 6: The maximum Fréchet distance (left) between a line segment (p_i, p_j) and its corresponding subpolyline is $\leq 2\delta$ if there is a valid shortcut (p_y, p_z) going over p_i and p_j. This is not true for the Hausdorff distance (right) where the distance between (p_i, p_j) and p_{i+1} can be arbitrarily large.

w segments, and hence we have $m \leq tw$. Applying the greedy algorithm to this instance gives an approximation ratio in $\mathcal{O}(\log(tw)) = \mathcal{O}(\log(t + w))$.[7]

It remains to prove the polynomial running time. Using a polyline simplification algorithm [34] for the local Fréchet distance independently for each polyline, we can find all (maximal) shortcuts for every vertex on every polyline in $\mathcal{O}(\ell n^2 \log n)$ time. Combining these shortcuts at every vertex gives us all n maximal stars in $\mathcal{O}(\ell n)$ time. For each star, we also store the number of segment–polyline pairs it covers and, to each segment–polyline pair, we link all stars it appears in. Both can be done in $\mathcal{O}(\ell n^2)$ time. As long as there are uncovered segment–polyline pairs, we find the star with the most uncovered segment–polyline pairs and then update the number of uncovered segment–polyline pairs for the other stars. This can be done in $\mathcal{O}(\ell n^2)$ time in total as well. ■

4.4 Relationship between Star Covers and Solutions of PBS

While a solution for PBS can be directly converted into a star cover as argued in Section 4.2, the converse is more intricate. The shortcuts contained in the stars of an optimal StCo solution may be overlapping or nested along a polyline, that is, vertices skipped by one shortcut may be endpoints of another shortcut in the star cover. Moreover, shared parts of multiple polylines may be covered by different stars. In other words, consistency is not guaranteed (e.g., in Figure 7d the shortcut of the purple star skips a vertex that is an endpoint of the other stars). We explain, however, how to derive from a star cover solution a (relaxed) solution for its corresponding instance of PBS. Some of the shortcuts of the StCo solution are replaced by "shorter" shortcuts in order to integrate some intermediate points to the PBS solution. Lemma 7 states that these newly introduced shortcuts can be at most 2δ away from the original polyline. The situation described there is depicted in Figure 6 for the Fréchet distance and – with a counterexample – for the Hausdorff distance. It follows immediately from a lemma by Agarwal et al. [1].

[7] Note that $\mathcal{O}(\log(tw)) = \mathcal{O}(\log(t + w))$ because of the following argument. As $t, w \geq 1$, $\mathcal{O}(\log(tw)) \supseteq O(\log(t + w))$ is clear. We next show that $\mathcal{O}(\log(tw)) \subseteq \mathcal{O}(\log(t + w))$. Suppose that $t \geq w$; the other case is symmetric. Then, $\mathcal{O}(\log(tw)) \subseteq \mathcal{O}(\log(t^2)) = \mathcal{O}(2\log(t)) = \mathcal{O}(\log(t)) \subseteq \mathcal{O}(\log(t + w))$.

Lemma 7 ([1], Lemma 3.3)**.** *Given a polyline $L = (p_1, p_2, \ldots, p_{|L|})$ and a distance threshold δ. If there are $y, z \in \mathbb{N}$ with $1 \leq y < z \leq |L|$ and $d_{\mathrm{F}}((p_y, p_z), L[p_y, \ldots, p_z]) \leq \delta$ (i.e., segment (p_y, p_z) is a valid shortcut), then for any $i, j \in \mathbb{N}$ with $y \leq i < j \leq z$, $d_{\mathrm{F}}((p_i, p_j), L[p_i, \ldots, p_j]) \leq 2\delta$.*

Equipped with this lemma, we now discuss the actual transformation from a StCo solution to a PBS solution with distance threshold 2δ. The idea is to keep, beside the first and last vertices of all polylines, only the central vertices of the selected stars while dropping their leaves. This is closely tied with the fact that we minimize the number of stars while ignoring their degree in the algorithm. The main insight here is that the shortcuts induced by this augmented vertex set still have a small distance to the original polylines.

Lemma 8. *Let C be a star cover for an instance of StCo obtained from an instance $(P, \mathcal{L}, \delta)$ of PBS under the local Fréchet distance. If C is an α-approximation for the instance of StCo, a bi-criteria $(\alpha+1, 2)$-approximation for $(P, \mathcal{L}, \delta)$ under the local Fréchet distance can be computed in $\mathcal{O}(n)$ time from C.*

Proof. Let P_{central} be the set of central vertices of the stars in C and let P_{end} be the set of first and last vertices of all polylines from $\mathcal{L}$. We return $P_{\text{central}} \cup P_{\text{end}}$ as the bi-criteria approximate solution of $(P, \mathcal{L}, \delta)$. Clearly, $P_{\text{central}} \cup P_{\text{end}}$ induce a simplified polyline bundle on the set of original polylines $\mathcal{L}$. We call this simplified set of polylines $\mathcal{L}'$. According to Lemma 5, $\mathsf{OPT}_{\text{StCo}} \leq \mathsf{OPT}_{\text{PBS}}$. We conclude

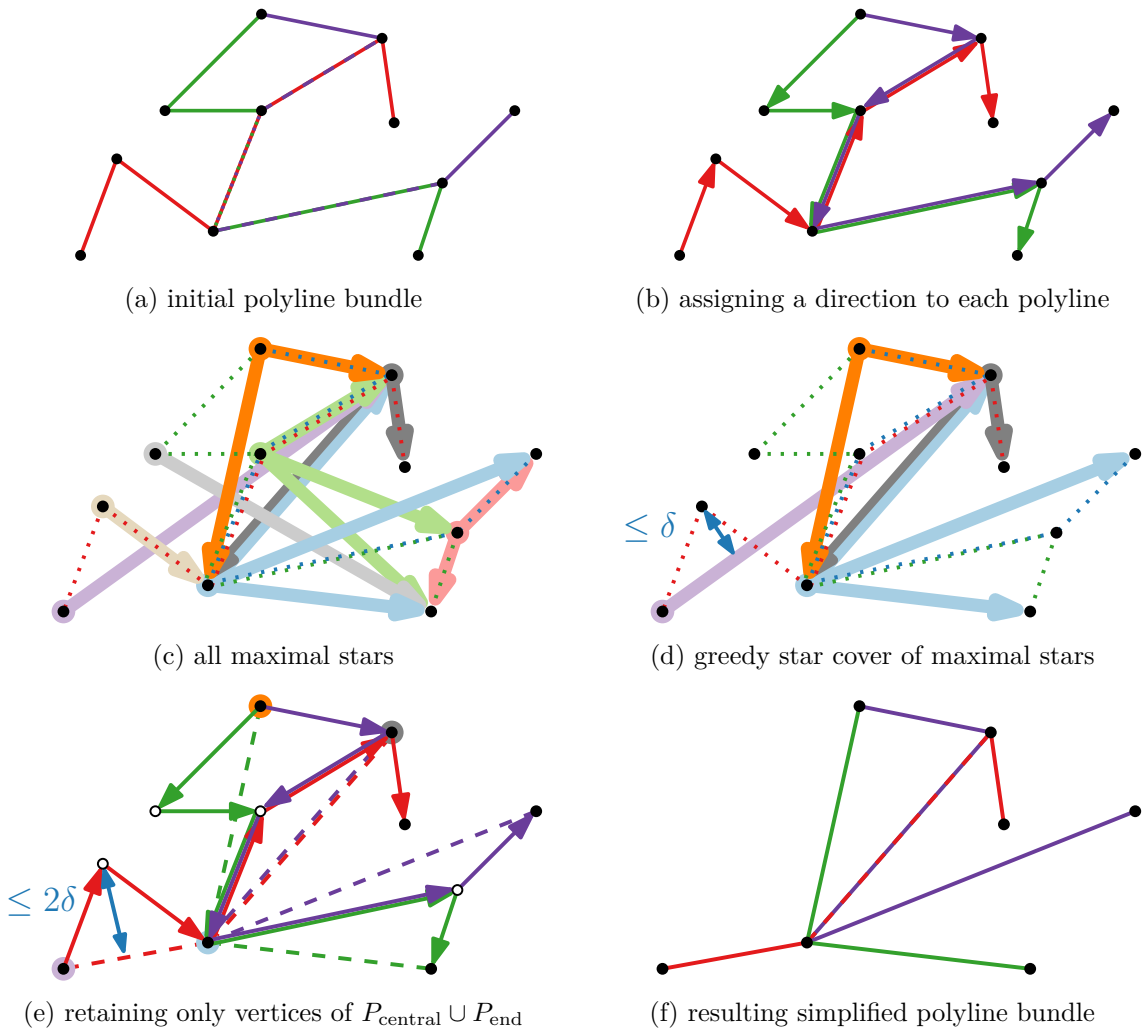
$$|P_{\text{central}} \cup P_{\text{end}}| \leq \alpha \mathsf{OPT}_{\text{StCo}} + \mathsf{OPT}_{\text{PBS}} \leq (\alpha + 1)\mathsf{OPT}_{\text{PBS}}\,. \tag{4}$$

It remains to prove that the local Fréchet distance between $\mathcal{L}'$ and $\mathcal{L}$ is at most 2δ. Consider any segment (p_i, p_j) of a (simplified) polyline $L' \in \mathcal{L}'$ corresponding to an (original) polyline $L \in \mathcal{L}$ such that p_i precedes p_j in (the directed version of) L. Notice that there is a single star s in C that covers all segments of $L[p_i, p_j]$. Otherwise, there would be another central vertex of a star between p_i and p_j on L and, in L', (p_i, p_j) would not be a segment. The central vertex p_{central} of s precedes p_i or is equal to p_i as otherwise s would not cover all of $L[p_i, p_j]$. Similarly, the outer vertex p_{outer} of s on L succeeds p_j or is equal to p_j as otherwise s would not cover all of $L[p_i, p_j]$. By the definition of a star, we know that $d_{\mathrm{F}}((p_{\text{central}}, p_{\text{outer}}), L[p_{\text{central}}, p_{\text{outer}}]) \leq \delta$. By Lemma 7, it follows that $d_{\mathrm{F}}((p_i, p_j), L[p_i, p_j]) \leq 2\delta$. ∎

4.5 Bi-criteria Approximation for PBS via StCo

We have now gathered all lemmas to obtain the main theorem of this section.

Theorem 2. *There is a bi-criteria $(\mathcal{O}(\log(\ell + n)), 2)$-approximation algorithm for PBS under the local Fréchet distance running in $\mathcal{O}(\ell n^2 \log n)$ time, where ℓ is the number of polylines and n is the number of vertices in the polyline bundle.*



(a) initial polyline bundle

(b) assigning a direction to each polyline

(c) all maximal stars

(d) greedy star cover of maximal stars

(e) retaining only vertices of $P_{\text{central}} \cup P_{\text{end}}$

(f) resulting simplified polyline bundle

Figure 7: Example of our bi-criteria $(\mathcal{O}(\log(\ell + n)), 2)$-approximation algorithm for PBS.

Proof. The steps described above provide an approximation-preserving reduction from PBS to StCo, which can be realized as a bi-criteria approximation algorithm. Its steps are depicted in Figure 7.

Given an instance $(P, \mathcal{L}, \delta)$ of PBS, where we let the size of an optimal solution be $\mathsf{OPT}_{\text{PBS}}$, we assign an arbitrary direction to each $L \in \mathcal{L}$ and construct the corresponding instance of StCo where we only store the maximal stars. For this corresponding instance of StCo, we compute an $\mathcal{O}(\log(t+w))$ approximation star cover C via the greedy approach described in Section 4.3. We can do this in $\mathcal{O}(\ell n^2 \log n)$ time according to Lemma 6. According to Lemma 8, we can use C to compute a bi-criteria $(\mathcal{O}(\log(t+w)), 2)$-approximation for $(P, \mathcal{L}, \delta)$ in $\mathcal{O}(n)$ time. Since $t \leq \ell$ and $w \leq n$, this is also a bi-criteria $(\mathcal{O}(\log(\ell+n)), 2)$-approximation. ∎

It is reasonable to assume that the number ℓ of polylines is polynomial in n in practically relevant settings. Hence, we essentially obtain an exponential improvement over the complexity-theoretic lower bound $n^{1/3-\varepsilon}$ if we allow a minor violation of the error bound. We remark that Theorem 2 does not carry over to the local Hausdorff distance since Lemma 7 does not hold for the Hausdorff distance.

5 Fixed-Parameter Tractability for Polyline Bundles of General Graphs

A brute force approach to solve PBS is checking for every subset of the vertex set P in $\mathcal{O}(\ell \cdot n)$ time whether it is a valid simplification and returning the one with the smallest number of vertices. Consequently, the runtime of this approach is $\mathcal{O}(2^n \cdot \ell \cdot n)$. We next present a simple approach improving this runtime to FPT time.

When considering fixed-parameter tractability (FPT), investigating parameters of the input is a natural choice. According to Corollary 2, PBS is not FPT in the number of polylines ℓ. However, PBS is FPT in the number of shared vertices, i.e., vertices contained in more than one polyline. We denote the set of those vertices by P_{shared} and we define $k := |P_{\text{shared}}|$.

Theorem 3. *PBS is fixed-parameter tractable (FPT) in the number of shared vertices k. There is an algorithm solving PBS in $\mathcal{O}(2^k \cdot \ell n^2)$ time under the local Hausdorff distance and in $\mathcal{O}((2^k + \log n) \cdot \ell n^2)$ time under the local Fréchet distance.*

Proof. We describe an algorithm that solves PBS in FPT time. Given an instance $(P, \mathcal{L}, \delta)$ of PBS, the first step is to compute, for each $L \in \mathcal{L}$, its shortcut graph G_L using a classical polyline simplification algorithm. For the local Hausdorff distance, we can do this in $\mathcal{O}(n^2)$ time [13], and for the local Fréchet distance, we can do this in $\mathcal{O}(n^2 \log n)$ time [34]. Hence, for all polylines of $\mathcal{L}$, we compute their shortcut graphs in time $\mathcal{O}(\ell \cdot n^2)$ and $\mathcal{O}(\ell \cdot n^2 \log n)$, respectively.

The second step is to iterate over all subsets $P' \subseteq P_{\text{shared}}$ and check if $P' = P_{\text{shared}} \cap P^\star$ where $P^\star$ is the vertex set of an optimal solution. Before the first iteration, we initialize a variable $n_{\min} = n$ and we will save the current best solution by $\mathcal{S}_{\min}$, which we initialize with $\mathcal{L}$. Then, in each iteration, we temporarily remove from all of our shortcut graphs all nodes $P_{\text{not-contained}} = P_{\text{shared}} - P'$ and all edges that correspond to a shortcut skipping a vertex in P'. Clearly, removing $P_{\text{not-contained}}$ can be performed in $\mathcal{O}(n^2)$ time for each shortcut graph G_L. For the removal of the edges in G_L, note that we can sort the list of vertices P' and the list of all edges (defined by their endpoints) lexicographically by the occurrence of the vertices within the polyline L. If we traverse both lists simultaneously in ascending order, we remove an edge if and only if its endpoints come before and after the currently considered vertex from P'. Therefore, the removal operations can be performed in $\mathcal{O}(n^2)$ time per G_L.

If some shortcut graph becomes disconnected by these removal operations, we continue with the next iteration. Otherwise, we take the vertices of a shortest path from the first to the last vertex in each reduced version of G_L. This yields a simplified polyline S_L

of L. Together they define a simplification $\mathcal{S}$ of our PBS instance. Observe that this simplification is consistent because each S_L contains, from the shared vertices P_{shared}, precisely the vertices in P'.

If the number $n_{\mathcal{S}}$ of vertices in $\mathcal{S}$ is less than $n_{\min}$, we set $n_{\min} = n_{\mathcal{S}}$ and $\mathcal{S}_{\min} = \mathcal{S}$. In the end, we return $\mathcal{S}_{\min}$. Since we have 2^k subsets of P_{shared} and each iteration can be performed in $\mathcal{O}(\ell \cdot n^2)$ time, the running time of the second step is in $\mathcal{O}(2^k \cdot \ell \cdot n^2)$.

It remains to prove that $\mathcal{S}_{\min}$ is an optimal solution of our input instance of PBS. The returned solution is valid because the shared vertices of P' are retained in all simplified polylines (they cannot be skipped) and the other shared vertices are skipped in all simplified polylines. Our algorithm finds the minimum size solution because in one iteration it considers $P' = P_{\text{shared}} \cap P^\star$. Moreover, an optimal solution cannot have fewer vertices occurring in only one polyline L than our algorithm since this would imply a shorter shortest path within the reduced version of G_L. ■

6 Polyline Bundles of Trees

With the general PBS problem being hard to approximate better than $n^{1/3}$ even on planar inputs, the question arises whether there are special cases that are better tractable. In this section, we will answer this positively by proving that the PBS problem can be solved efficiently on tree bundles. Furthermore, we also describe how reasonable solutions on general bundles can be obtained by first decomposing them into tree bundles and then simplifying each tree bundle optimally.

6.1 Formal Definition

To form a tree bundle, the polylines have to start in a common root point and then branch out. This happens naturally, for instance, in river networks. Figure 8, left, shows an example of a tree bundle.

Definition 8 (Polyline Tree Bundle (PTB))**.** *A* polyline tree bundle (PTB) *is a PBS instance* $(P, \mathcal{L}, \delta)$ *where we additionally require that* $\mathcal{L}$ *is a set of simple polylines such that all* $L \in \mathcal{L}$ *start at a common root vertex* p_r*, and for any pair of polylines* $L, L' \in \mathcal{L}$*, the only intersection is a common prefix* $L[p_r, p_i] = L'[p_r, p_i]$*.*

We exclude the case that a polyline $L' \in \mathcal{L}$ is a subpolyline of another polyline $L \in \mathcal{L}$. By the definition of the PBS problem, we include the endpoints of all polylines in our simplification and hence if the endpoint of L' lies on L, we could simply consider that point as the root of another PTB, which can be simplified independently. Further, we want to emphasize that a common prefix as defined above demands the polylines to start with same set of polgon vertices. Thus, if one polyline would contain an additional node in the middle of the edge of another polyline, the two subpolylines would not be considered as a common prefix according to our definition despite having the same shape. We remark, though,

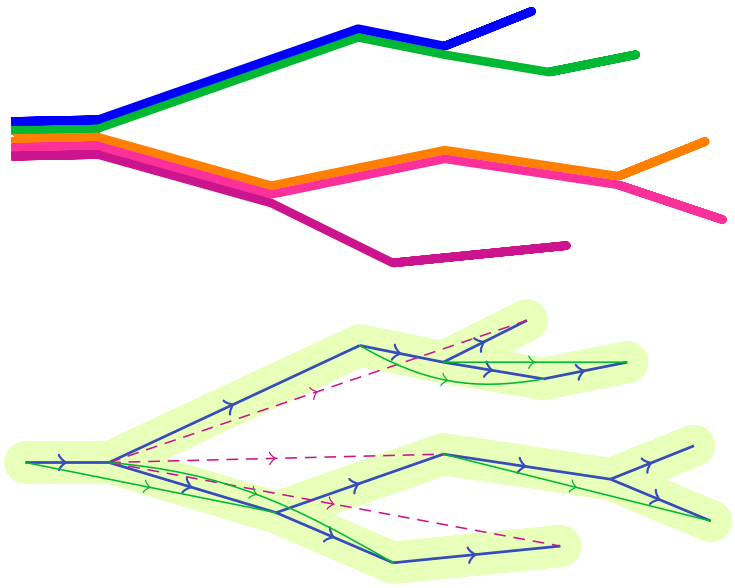


Figure 8: Top: Example of a PTB instance. Bottom: Thick blue edges represent the tree graph G_t. The combination of the thick blue and the green edges constitutes the shortcut graph G_s for d_F for the distance threshold δ (indicated via the light green tubes). Examples of invalid shortcuts are drawn dashed violet.

that the PTB definition does not demand the tree bundle to be planar as the intersection constraint is only concerned with common points of the polylines.

For simplification of single polylines, the Imai–Iri algorithm [27] first constructs a shortcut graph and then computes the shortest path therein to find the optimal result. We will proceed in a similar fashion by first transforming the PTB simplification problem into a graph problem and then computing the optimal solution based on this. However, to more efficiently deal with the tree structure, the graph construction as well as the final search for the best simplification is more intricate. In particular, we will use two different types of graphs in our transformation, and the shortest path search will be replaced by a suitable dynamic programming approach.

6.2 Transformation into a Graph Problem

We transform the PTB simplification problem into a graph problem by constructing two directed graphs from the input data: a *tree graph* and a *shortcut graph*. We start by considering the polylines as embedded directed paths which start at the common root p_r. The tree graph $G_t = (V, E_t)$ is the union of these paths. More precisely, for each vertex occurring in the PTB, there is a corresponding node $v \in V$ (with v_r corresponding to the root vertex p_r), and there exists a directed edge $(v, w) \in E_t$ if there is a polyline $L \in \mathcal{L}$ which contains the segment between the respective vertices (in that direction). For a given distance function d and threshold $\delta > 0$, the shortcut graph $G_s = (V, E_s)$ is the union of

all valid shortcut edges, i.e., there is an edge $(v, w) \in \binom{V}{2}$ if for every polyline $L \in \mathcal{L}$ that contains v and w (in that order), we have $d((v, w), L[v, w]) \leq \delta$. Figure 8 shows G_t and G_s for an example PTB. Note that no matter the distance function and the value of δ, we always have $E_\mathrm{t} \subseteq E_\mathrm{s}$, i.e. all tree graph edges are also contained in the shortcut graph.

Lemma 9. *The tree graph $G_\mathrm{t} = (V, E_\mathrm{t})$ has size $\mathcal{O}(n)$ and can be constructed in $\mathcal{O}(n^2)$ time.*

Proof. As all polylines in a PTB start at the same root vertex and end in one of the at most n leaf vertices, there can be at most n polylines in total. Each of these polylines consists of at most n segments itself, as all of them are required to be simple. Thus, the input polyline bundle contains $\mathcal{O}(n^2)$ segments.

The union of the directed paths induced by the polylines can then be constructed by considering the polylines one-by-one. For a polyline, we start at its non-root endpoint and then add the edges that represent the respective polyline segments backwards one after the other until we either reach the root node or a node which already was considered as a segment endpoint in another polyline (then clearly the remaining path is already part of G_t). This process takes $\mathcal{O}(n^2)$ time and, given the tree structure, leads to an edge set E_t with $|E_\mathrm{t}| \in \mathcal{O}(n)$. ∎

Theorem 4. *The shortcut graph $G_\mathrm{s} = (V, E_\mathrm{s})$ has size $\mathcal{O}(n^2)$ and can be constructed for the local Hausdorff distance in $\mathcal{O}(n^2)$ time and for the local Fréchet distance in $\mathcal{O}(n^2 \log n)$ time.*

Proof. We compute the set of valid shortcuts by considering the polylines one-by-one in an arbitrary order. To avoid redundant computations along shared parts, we store the result for already considered node pairs. Accordingly, the total number of potential shortcuts that need to be checked is in $\mathcal{O}(n^2)$. The time T_d to check the validity of a shortcut is in $\mathcal{O}(n)$ for both d_F and d_H. Therefore, the total construction time would be in $\mathcal{O}(n^3)$.

However, for d_H we can do better by leveraging the sweep method from Chan and Chin for shortcut computation for single polylines in $\mathcal{O}(n^2)$ time [13]. But we cannot apply that method to each polyline individually, though, as then the running time would be again in $\mathcal{O}(n^3)$. Hence to get an improvement, we have to avoid redundant computations along shared parts. We achieve that by sweeping the induced tree instead of the individual polylines. As in the original Chan and Chin algorithm, this requires a forward sweep and a backward sweep: We first consider the shortcuts directed towards the root of the tree bundle. Using the sweep algorithm, we consider at most n vertices as starting points and for each vertex there are at most n vertices on the unique path that connects the point to the root. Hence all shortcuts pointing towards the root can be computed in $\mathcal{O}(n^2)$ time. In the backward direction, we exploit the tree graph G_t as follows. For each vertex p, we compute the shortcuts starting at p by exploring the subtree rooted at v_p in G_t in a depth-first search (DFS) manner. For each branching point in the tree, we make a copy of the cone which describes the possible set of shortcut endpoints and operate on that copy in the respective subtree. Apart from that, we proceed exactly as in the classical Chan and Chin

algorithm. As each subtree contains at most n nodes and as we need at most one copy of the region at any point, we can indeed compute all shortcuts starting at p in time linear in the total number of points. Hence also the set of shortcuts pointing away from the root can be computed in $\mathcal{O}(n^2)$ time.

For the local Fréchet distance, sweeping only towards the root suffices to determine all shortcuts in $\mathcal{O}(n^2 \log n)$ time [34]. In conclusion, we can compute the shortcut graph for a given PTB in time $\mathcal{O}(n^2)$ and $\mathcal{O}(n^2 \log n)$ for the local Hausdorff and the local Fréchet distance, respectively. ∎

Based on the notion of the tree graph and the shortcut graph, we are now ready to restate the PTB simplification problem (PTBS) as a graph problem.

Definition 9 (Polyline Tree Bundle Simplification (PTBS))**.** *Given a tree graph $G_\mathrm{t} = (V, E_\mathrm{t})$ and a shortcut graph $G_\mathrm{s} = (V, E_\mathrm{s})$, the objective is to find a minimum-size node subset $S \subseteq V$ such that:*

- *The root node and all leaf nodes of the tree graph G_t are contained in S.*
- *The induced subgraph $G_\mathrm{s}[S]$ is connected.*

6.3 An Exact Polytime Algorithm

Next, we describe a dynamic programming (DP) approach that only operates on G_t and G_s, and returns an optimal PTBS solution in $\mathcal{O}(n^2)$ time. Let $\mathsf{Sub}(v) \subseteq G_\mathrm{t}$ be the sub-tree rooted at node v in the tree graph. Our main observation is that we can break down an optimal solution recursively. If a node v is part of the solution, it's easy to see that there can't be shortcuts bypassing v. Thus, the solution S can be split into two parts: an optimal solution for $\mathsf{Sub}(v)$ and an optimal solution for $G_\mathrm{t} \setminus \mathsf{Sub}(v)$. We denote the size of an optimal solution for $\mathsf{Sub}(v)$ by $s(v)$.

As we don't know a priori which nodes will end up in the solution, we strive for computing $s(v)$ for each node $v \in V$ in an efficient manner. For every leaf node v, we obviously get $s(v) = 1$. To compute $s(v)$ for an inner node v, we assume that $s(w)$ is already known for every node $w \in \mathsf{Sub}(v) \setminus \{v\}$. Each path from a leaf u to v in $\mathsf{Sub}(v)$ needs to contain a *cover node* w such that $(v, w) \in E_\mathrm{s}$ (that means there is a valid shortcut between v and w). To identify the best selection of such cover nodes, we compute a helping function $h\colon V \to \mathbb{N}$ for each node $w \in \mathsf{Sub}(v)$ as follows: Initially, $h(w) = s(w)$ if $(v, w) \in E_\mathrm{s}$, and $h(w) = \infty$ otherwise. Then, in a post-order traversal of $\mathsf{Sub}(v)$, for each non-leaf node w we set $h(w) = \min\{h(w), \sum_{u \in N(w)} h(u)\}$ where $N(w)$ denotes the set of children (out-neighbors) of w in G_t. In that way, $h(w)$ encodes the smallest number of nodes that have to be kept in $\mathsf{Sub}(w)$ if for all paths from v to leaf nodes in $\mathsf{Sub}(w)$ the respective cover node is contained in $\mathsf{Sub}(w)$. The optimal solution size $s(v)$ for $\mathsf{Sub}(v)$ is then $h(v) + 1$ (as we have to additionally include v itself).

Note that $s(v)$ is always well-defined (i.e., finite) as the tree edges are all valid shortcuts in G_s. To make sure that at the time we compute $s(v)$, all values $s(w)$ for

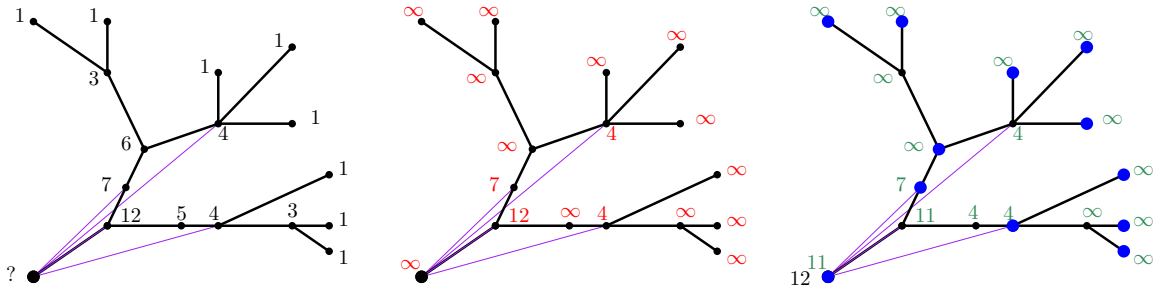


Figure 9: The left image shows an example tree graph with optimum sub-tree simplification sizes (black) known for all nodes except the root node. The purple line segments indicate valid shortcuts from the root node (note that these were chosen manually for the example and not based on the underlying geometry of the depicted tree). The middle image depicts the same tree after initial assignment of the helping values (red). Here, only end nodes of valid shortcuts have finite values assigned to them. The right image shows the final helping values (green) after propagation as well as the respective optimum simplification size of the tree assigned to the root node (black). The blue marked nodes are the ones that are contained in the optimal simplification.

$w \in \mathsf{Sub}(v) \setminus \{v\}$ are known, we also globally traverse the nodes in the tree graph in post-order. Figure 9 illustrates the computation of $s(v)$. The optimal set of simplification nodes can then be determined by backtracking.

For a faster running time of the DP in practice (used in our experiments), we only compute h-values for nodes in $\mathsf{Sub}(v)$ which are on a path from v to some node w with $(v, w) \in E_\mathrm{s}$. These nodes can easily be identified by computing the reverse path from each such node w to v and marking all nodes along the way (stopping as soon as a marked node is encountered to avoid redundancy). For marked nodes w with an unmarked neighbor, we just set $\sum_{u \in N(w)} h(u)$ to ∞ to maintain correctness. Especially for small distance thresholds δ and large sub-trees $\mathsf{Sub}(v)$, this modification accelerates the computation of $s(v)$ significantly.

Theorem 5. *PTBS can be solved optimally in $\mathcal{O}(n^2)$ time.*

Proof. Correctness can be shown by structural induction. The induction hypothesis is that a node v gets assigned the optimal simplification size $s_{\mathsf{OPT}}(v)$ for its sub-tree assuming v is kept in the solution. For leaf nodes (that get assigned a value of 1), correctness is obvious. Now we consider some non-leaf node v and assume that all nodes w in the respective sub-tree received correct solution size values, that is, $s(w) = s_{\mathsf{OPT}}(w)$. Let C be the set of outgoing shortcut edges emerging from v in an optimal simplification of $\mathsf{Sub}(v)$, leading to an induced simplification size of $s_{\mathsf{OPT}}(v) = 1 + \sum_{(v,w) \in C} s(w)$. In the DP, each node w with $(v, w) \in C \subseteq E_\mathrm{s}$ gets assigned the helping value $h(w) = s(w)$ when processing v. Based on the propagation of the h-value towards the root node, we observe that $h(v) \leq \sum_{(v,w) \in C} s(w)$ and therefore $s(v) \leq 1 + \sum_{(v,w) \in C} s(w) = s_{\mathsf{OPT}}$. We can never get $s(v) < s_{\mathsf{OPT}}(v)$ as the value assigned to v always represents a valid simplification of $\mathsf{Sub}(v)$. Hence we conclude that $s(v) = s_{\mathsf{OPT}}(v)$.

The time to compute $s(v)$ is in $\mathcal{O}(|\mathsf{Sub}(v)|+|\{(v,w)\in E_s\}|)$ as it requires to consider all nodes in the subtree rooted at v (in post-order) and to check all of its out-neighbors in G_s. Both of these sets have a size that is upper bounded by the number of nodes in $\mathsf{Sub}(v)$, and the post-order can be determined in $\mathcal{O}(|\mathsf{Sub}(v)|)$ using a DFS run. Accordingly, the respective computation time is in $\mathcal{O}(n)$ for each v and in $\mathcal{O}(n^2)$ for all nodes $v\in V$ combined. The post-order traversal to determine the global order in which the nodes are processed takes only linear time in n (again using a DFS run). Hence altogether, we have two nested post-order traversals with a total running time of $\mathcal{O}(n^2)$. ∎

We note that the correctness and the running time of the DP approach do not depend on the used distance measure for shortcut creation. In fact, the algorithm works with any shortcut graph G_s as long as it contains the tree G_t as a subgraph. Therefore, the shortcut graph could also come from other means of construction (including customization). We further remark that minimizing the number of vertices in the simplified tree bundle is equivalent to minimizing the number of segments, while for general polyline bundles these two optimization goals might lead to different results.

Combining the time for problem transformation with the time of the DP, we get an overall running time of $\mathcal{O}(n^2\log n)$ for PTBS when using the Fréchet distance and a running time of $\mathcal{O}(n^2)$ when using the Hausdorff distance. Hence – although having to use more complicated machinery – we end up with running times for tree bundle simplification that match the best known running times for simplification of a single polyline.

6.4 Decomposition of General Bundles into Tree Bundles

To leverage our algorithm for optimal tree bundle simplification for general bundles, we next consider the problem of decomposing a general bundle into (a small set of) tree bundles. To formalize the tree bundle decomposition (TBD) problem, we first introduce the notion of a D-decomposition of a polyline.

Definition 10 (D-Decomposition)**.** *Let $L=(s,\dots,t)$ be a simple polyline (represented as a list of points) and let D be a point set. Further let $d_1,d_2,\dots,d_k$ be the points in $L\cap D$ in the order in which they appear in L. The D-decomposition of L denoted by $L(D)$ is the set of subpolylines $L[d_i,d_{i+1}]$ for $i=1,\dots,k-1$.*

We strive to find a point set D such that its induced D-decomposition of the input bundle provides us with a (small) set of tree bundles.

Definition 11 (Tree Bundle Decomposition (TBD))**.** *Given a PBS instance $(P,\mathcal{L},\delta)$, we seek to find a point subset $D\subseteq P$ (the decomposition points) with the following requirements:*

- *Each polyline $L\in\mathcal{L}$ starts and ends in a point in D.*
- *Let G_I be the intersection graph in which we have a node for each subpolyline in $\bigcup_{L\in\mathcal{L}}L(D)$ and an edge between two nodes if the subpolylines $\tilde{L}$ and $\bar{L}$ share a point that is not in D, i.e. $(\tilde{L}\cap\bar{L})\setminus D\neq\emptyset$. Then the subpolylines within a connected component in G_I form a PTB.*

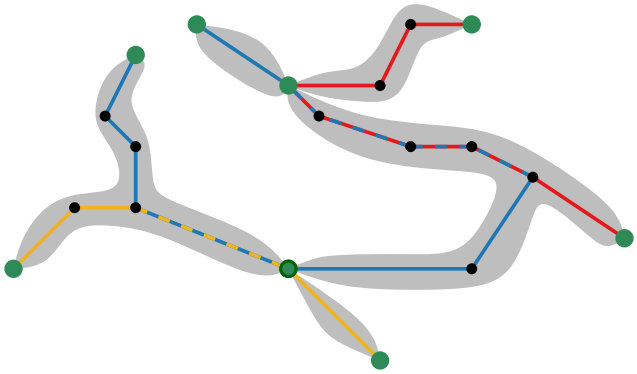


Figure 10: Input polyline bundle (same as in Figure 1) and decomposition set D (green). The D-decomposition is a valid tree bundle decomposition. The tree bundles are indicated by the gray regions. Note that points in D may be the root of one tree bundle and a leaf of another one at the same time. This applies, for example, for the green node with the darker green boundary, which is the root of the yellow-blue-bundle to its left and a leaf of the blue-red-bundle to its right.

Figure 10 shows an example of a valid TBD. Based on a TBD, we can simplify the given bundle by simplifying each of the tree bundles induced by D independently. The union of all tree simplifications then yields the set of retained vertices $S \subseteq P$. The goal, of course, is still to end up with a small set S. To achieve that, we aim at TBDs which induce few but large tree bundles with a small decomposition set D. Nodes in the set D might end up being the root node of a tree bundle or a leaf node (or both). In the following we assume that all polyline endpoints are already included in D as they have to be part of S by definition.

To guide the decomposition, we construct a union graph $G_U(V, E)$ of the input polyline bundle. Here, each vertex in the bundle is represented by a node in V and an edge exists between two nodes if there is a polyline segment between the respective points. Additionally, we assign to each edge in G_U the set of polylines that traverse it.

6.4.1 Path Bundle Decomposition

The most simple type of tree bundle decomposition is to restrict the resulting bundles to be path bundles, i.e., no branching is allowed to occur. Hence, when considering G_U all nodes $v \in V$ with a degree $deg(v) \neq 2$ have to be included in D. Accordingly, the respective decomposition can be computed in linear time in the size of G_U, and for the resulting path bundles, classical polyline simplification algorithms can be applied individually. Figure 11, left, shows a small example.

6.4.2 Greedy Tree Bundle Decomposition

To end up with a decomposition not only consisting of path bundles but general tree bundles, we now propose a greedy approach. In particular, the idea is to greedily select root nodes

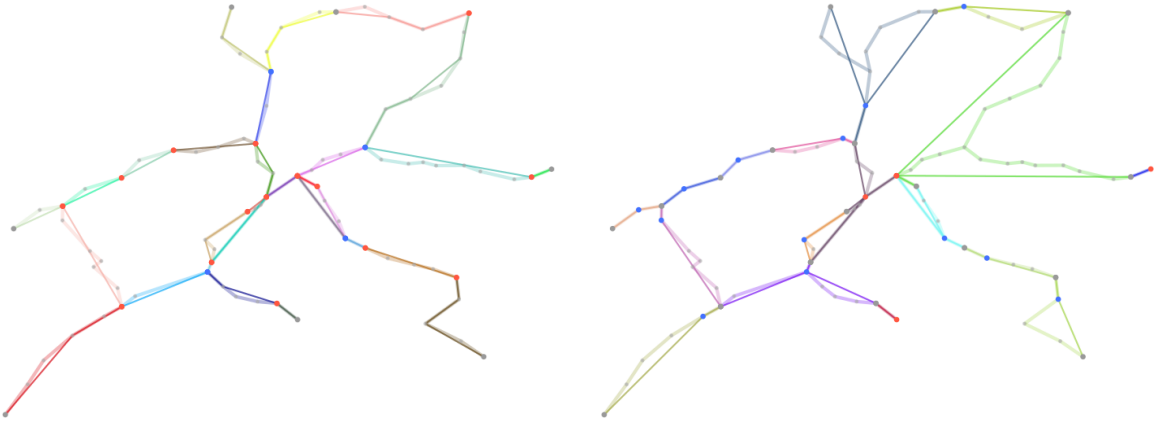


Figure 11: Path decomposition (left) and tree decomposition (right) and the resulting simplifications (with different ε values) of a public transit network of Stuttgart. The original polylines are drawn with different colors with reduced opacity as a base layer. On top, the decomposition set D is shown (large nodes, colors are based on their roles as root or leaf node or multi-purpose), as well as the inserted shortcuts.

and grow trees from those in G_U (adding the respective leaf nodes to D as well).

For root selection, we use the line degree of a vertex, that is, the number of polylines in $\mathcal{L}$ that contain the vertex. As only polylines that contain the root can be part of the respective PTB, we always choose the node with the highest line degree that is not already part of a tree bundle next. In the example in Figure 10, the maximum line degree is 2 and choosing any of the vertices which have two incident lines would be fine.

The largest possible tree bundle for a selected root node r can be computed by a kind of a breadth-first search (BFS) in G_U. Starting from r, we always push edges instead of nodes in the queue. The edges incident to r are always included in the PTB and are hence used for initialization of the queue (artificially directed away from r). In any later step, if an edge (u, v) is extracted from the queue, we first check whether all other edges incident to v are unvisited. If that is the case, we need to make sure that the polyline set assigned to each incident edge is a (not necessarily proper) subset of the polylines assigned to (u, v). If and only if those conditions are met for all incident edges, these edges are included in the subtree, marked as visited, and inserted into the queue. Otherwise v is added to D. The process takes $\mathcal{O}(\ell n)$ time since $\mathcal{O}(\ell n)$ is an upper bound for the number of edges in G_U and checking whether a set of polylines is a subset of another set of polylines takes $\mathcal{O}(\ell)$ time, which we do $\mathcal{O}(n)$ times. Figure 11, right, shows the resulting decomposition for a small public transit network.

6.4.3 Bottom-up Tree Decomposition

Next, we present a slightly different greedy heuristic. Instead of picking a root node and growing the tree from top to bottom, we start from the leave nodes and grow the trees upwards. Whenever two trees collide, we try to merge them to a single tree if possible.

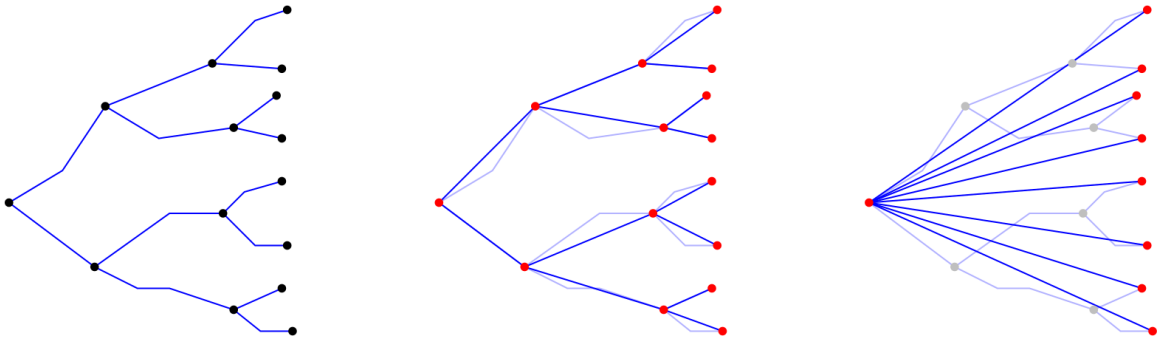


Figure 12: Binary tree (left) with path-decomposition based simplification (middle) and tree-decomposition based simplification (right) for unbounded distance threshold δ.

The idea of this approach is to defer the choice of root nodes to a later point in time. Notably, this *bottom-up* approach tends to create more trees, but it reduces the number of additional fixed nodes (root nodes). It turns out to be most effective in combination with a post-processing step described in Section 6.4.5.

6.4.4 Quality of Decomposition-based Simplification

The tree bundle decomposition is more involved and time-consuming than the path bundle decomposition. However, as we will show in the experimental evaluation, the tree bundle decomposition step takes negligible time compared to the subsequent simplification step. Furthermore, a simplification based on tree bundles might contain only half of the nodes of the simplification based on a path bundle decomposition. To see this, consider a complete binary tree of depth k. Then the path bundle decomposition based simplification has to keep all $2^{k+1}-1$ vertices for any choice of δ. The tree bundle based simplification, however, only needs to keep the root node and the leaves for a sufficiently large distance threshold δ, that is, 2^k+1 nodes. The ratio of those two terms converges to 2 for $k \to \infty$. Figure 12 illustrates the described setup for $k = 3$. Accordingly, a tree bundle decomposition can result in greatly improved simplifications than a path decomposition if the input polyline bundle contains tree-like substructures.

Nevertheless, both the path and the tree bundle decomposition based simplification are heuristic in nature. As can be seen in Figure 13, choosing one node to be contained in the solution might increase the resulting simplification size by a factor in $\Theta(n)$. However, in practical inputs, we don't expect such scenarios to arise frequently. If we decide to keep a node v in the simplification where for any shortcut (u, w) that spans it, also the edges (u, v) and (v, w) are contained in the shortcut graph, then the total size of the optimal simplification might only be increased by the number of polylines such a node v is contained in.

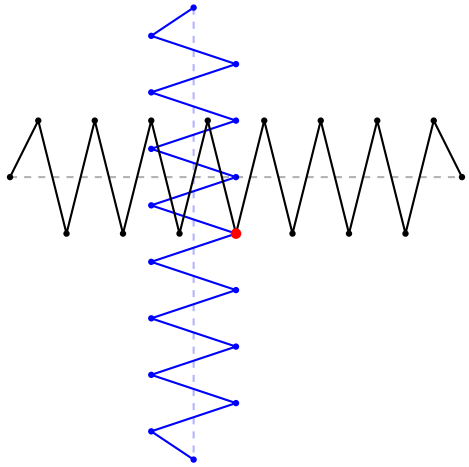


Figure 13: Example instance illustrating that deciding to keep a single selected vertex in the simplification might heavily impair the quality of the simplification: The optimal solution for the given bundle for suitable choice of δ would be to take the two dashed shortcut segments. However, when the marked red vertex has to be kept in the simplification, then – for a sufficiently small choice of δ – there are no valid shortcuts anymore. Then, the simplification has to contain all input vertices.

6.4.5 Post-processing Simplification Results

Tree bundle decomposition necessarily introduces fixed nodes that will be included into the final simplification result. Depending on the simplification threshold δ, this may be overly cautious. In a post-processing stage, we test for each fixed node in D, whether it can be omitted from the simplification without violating the threshold δ. It turns out that such a post-processing heuristic is able to improve the simplification results further, depending on the choice of δ.

We suppose this post-processing heuristic is particularly useful for grid-like structures that are commonly found in road or public transit networks. An example is depicted in Figure 14. Grid crossings are likely candidates for tree roots during tree decomposition (the large green dot in the center of the image on the left). However, these nodes are usually very close to polylines and can often be eliminated later from the solution without violating the threshold δ (image on the right).

7 Experimental Evaluation

We have implemented the bi-criteria approximation algorithm (BCA) from Section 4 as well as the DP approach for exact tree bundle simplification from Section 6.3 and the greedy tree bundle decomposition algorithm (TBD) from Section 6.4 in C++. As the quality and runtime assessments for BCA only hold when using the Fréchet distance, we restrict our experiments to the local Fréchet distance d_{lF}. All experiments were run on a single core of an Intel Core i9 processor at 2.4 GHz.

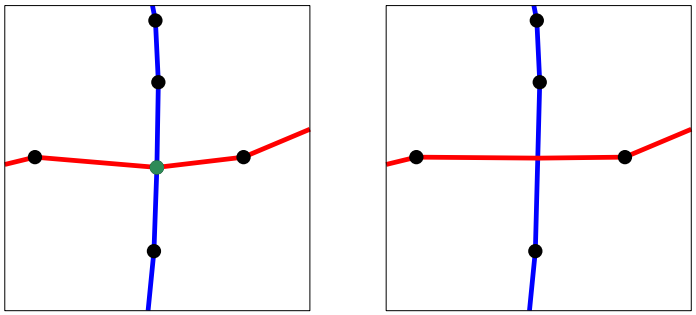


Figure 14: Post-processing heuristics on a grid-like graph. Given two polylines that intersect as shown in the left image, our tree-decomposition demands to include the green point in the center into the set D and thus to maintain it in the simplification. However, as shown on the right, deleting this point might still result in a valid simplification and so checking for ach point in D whether it can be pruned from the solution can greatly improve the simplification size.

7.1 Benchmark Data

We used two types of polyline bundle data to evaluate the algorithms:

(i) Path bundles from embedded road networks (extracted from OpenStreetMap [32]). Such bundles are a good model for movement data. Bundles were constructed by first extracting a connected subgraph with a given number of nodes from the network. To obtain a tree bundle, we then performed a BFS run from a randomly selected root node in the subgraph and backtracked all paths from the leaves to the root of the BFS-tree. For general bundles, we select not one but several root nodes in the subgraph, construct a tree bundle for each and then combine those into a single bundle. We suggest that such bundles represent likely movements of individuals or small groups of individuals in the network, e. g. from their homes to places of interest, quite well. Also note that a union of tree bundles cannot simply be decomposed into the respective trees again in case line segments are shared among them.

(ii) Public transit networks (GTFS data provided by OpenMobilityData [31]). We used the data from Stuttgart, Freiburg, Manhattan and Chicago. Here, each bus or train line constitutes a polyline in our bundle. Stations that are visited by multiple lines are then shared vertices and if consecutive stations are visited by multiple lines, the respective segment is shared.

7.2 Results on Tree Bundles

We compare the performance of DP and BCA on tree bundles of different sizes extracted from road networks. While it might seem to be an unfair comparison, if we have an exact algorithm on the one side and a bi-criteria approximation algorithm on the other side, it is not a priori clear which algorithm would produce the smaller simplification when tested with the same distance threshold δ (as BCA is allowed to exceed δ by a factor of 2). We

	$\delta \cdot 10^4$	$\delta_F \cdot 10^4$	δ_F/δ	n	$\lvert S\rvert$	time
DP	5.00	4.99	0.99	500	**204**	**3**
BCA	5.00	9.81	1.96	500	216	**3**
BCA	2.50	3.17	1.27	500	251	6
DP	5.00	5.00	1.00	8,000	**4009**	**21**
BCA	5.00	8.77	1.75	8,000	4029	407
BCA	2.50	4.04	1.61	8,000	5276	350
DP	5.00	5.00	1.00	50,000	**24,076**	**248**
BCA	5.00	9.68	1.94	50,000	24,195	14,800
BCA	2.50	5.00	2.00	50,000	32,457	13,500

Table 1: Comparison of DP and BCA on tree bundles (note that BCA is tested for the original δ and $\delta/2$). δ_F denotes the resulting Fréchet distance (applied with geo coordinates) and δ_F/δ the distance relative to the threshold. Furthermore, n is the number of vertices in the input and $|S|$ is the number of retained vertices in the computed solution. Timings are in milliseconds.

observe, however, that on all tested instances, the exact DP algorithm produces better simplification results than BCA, even though BCA is allowed to use a distance threshold of 2δ. If we call BCA with $\delta/2$ to end up with a solution that obeys the δ-constraint, the quality deteriorates significantly (with up to 50% larger outputs). Table 1 provides some selected results which reflect the general behavior. It is interesting to note that the BCA algorithm indeed produces solutions where the δ threshold is violated by a factor of 2, proving the theoretical analysis to be tight in this respect. We also observe that the DP approach scales much better, with running times up to a factor of 50 faster than BCA on our largest test instance.

In Figure 15, an example tree bundle instance is depicted along with DP results for a broad range of test instances. It can be observed that the optimal number of vertices in the simplification converges for growing values of δ to the number of leaf nodes in the tree graph, as those have to be kept by definition. But already for small δ, the simplification size comes close to that lower bound.

7.3 Tree Bundle Decomposition Results

Next, we evaluate the greedy tree bundle decomposition algorithm on public transit networks and road network path bundles. Unsurprisingly, the approach performed well on inputs with large tree-like structures as Freiburg and Stuttgart, but worse on instances as Manhattan or Chicago with large grid-like substructures.

For example, the public transit network of Stuttgart with 83 vertices was decomposed into 12 trees, see Figure 16, top. The decomposition set consists of 24 vertices of which 22 are endpoints of an input polyline and hence have to be included in the simplification anyway. Then, the optimal simplification of these trees for $\delta = 0.01$ has only 12 additional vertices.

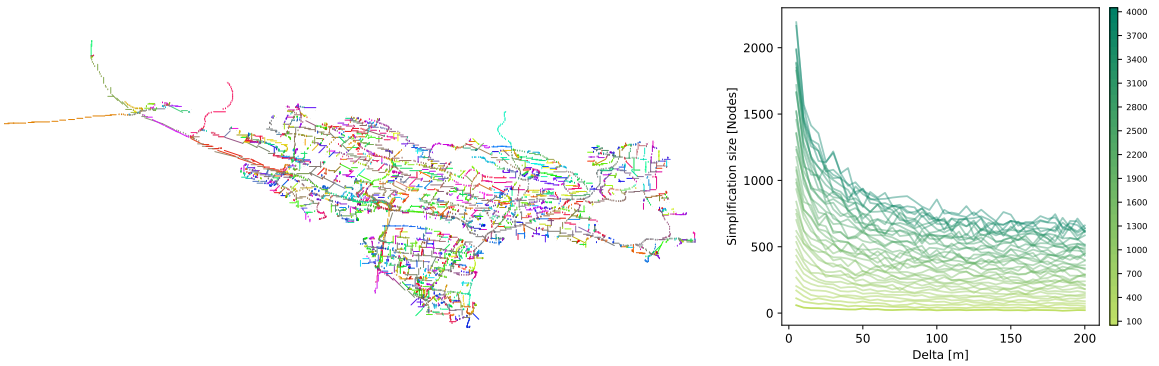


Figure 15: Left: Example of a road network tree bundle with 1365 polylines. Right: Number of retained vertices in an optimal simplification for tree bundles of different input sizes indicated by line color (legend on the right side) and distance thresholds (on the x-axis). The largest δ was chosen so that no further simplification is possible. Hence the respective simplification size for $\delta = 200\,\mathrm{m}$ (the right side of each curve) corresponds to the number of polylines in the input. One can nicely see that already for relatively small δ (around 50 m), the size of the simplification is close to this minimum.

The public transit network of Freiburg with 774 vertices was decomposed into 178 trees, see Figure 16, bottom. Here, 202 vertices in the decomposition originate from line endpoints and only 57 additional root nodes were selected. The simplification of the trees then added 92 nodes for $\delta = 0.002$ and 321 nodes for $\delta = 0.0002$. Post-processing heuristics, as desribed in Section 6.4.5, were able to reduce the number of additional nodes to 64 (for $\delta = 0.002$). For smaller values of δ the post-processing heuristics proved to be less effective.

The public transit network of Chicago with 9,984 points was decomposed into 2,649 trees, see Figure 17. Essentially, many crossing points in the grid structure were added to D, resulting in a limited simplification capability of the resulting trees.

Table 2 show the results for our tree-decomposition heuristics and the effectiveness of our post-processing. The 'top-down' tree-decomposition strategy (TD) tends to create fewer trees, but more fixed vertices. Conversely, the 'bottom-up' strategy (BU) is able to reduce the number of fixed vertices at the expense of creating more trees. In any case, the post-processing step from Section 6.4.5 does improve the simplification results, but most notably when combined with 'bottom-up' tree-decomposition.

7.4 Results on General Bundles

Finally we apply both TBD+DP and BCA to general bundles. Again, BCA results are allowed to exceed the distance threshold δ by a factor of 2. This slack is indeed strongly exploited also on public transit networks as confirmed by our experiments. We now focus on a comparative evaluation. We observe that our heuristic approach (bottom-up tree-decomposition with post-processing) of first computing a tree decomposition and then simplifying the resulting trees individually is always faster than BCA, computing results

	δ		trees	fixed	$\lvert S_0\rvert$	$\lvert S\rvert$
Freiburg	0.004	TD	178	259	324	281
		BU	236	184	332	**240**
	0.002	TD	178	259	351	320
		BU	236	184	352	**278**
	0.001	TD	178	259	408	389
		BU	236	184	375	**327**
Chicago	0.004	TD	2,649	3,013	4,544	3,663
		BU	3,308	1,711	4,270	**2,508**
	0.002	TD	2,649	3,013	4,602	3,827
		BU	3,308	1,711	4,317	**2,684**
	0.001	TD	2,649	3,013	4,837	4,289
		BU	3,308	1,711	4,447	**3,110**

Table 2: Effectiveness of tree-decomposition heuristics on public transit networks. TD='top-down', BU='bottom-up' heuristics; number of decomposed *trees*; number of *fixed* vertices; solution size before ($\lvert S_0\rvert$) and after ($\lvert S\rvert$) post-processing.

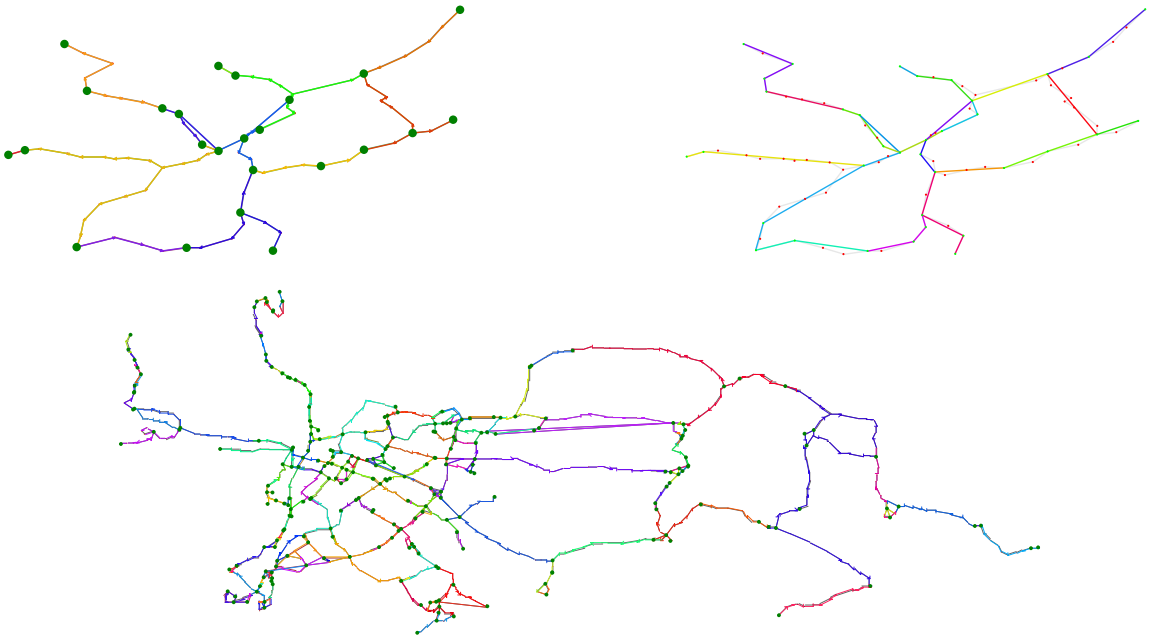


Figure 16: Tree decompositions of public transport networks computed with our (top-down) greedy heuristic. Large green points indicate the nodes in the decomposition set D. Each bus and train line received its own random color. Upper row: Decomposition result for the Stuttgart network on the left and resulting simplification on the right (with the original network in the background for comparison). Lower image: Decomposition result for the Freiburg network.

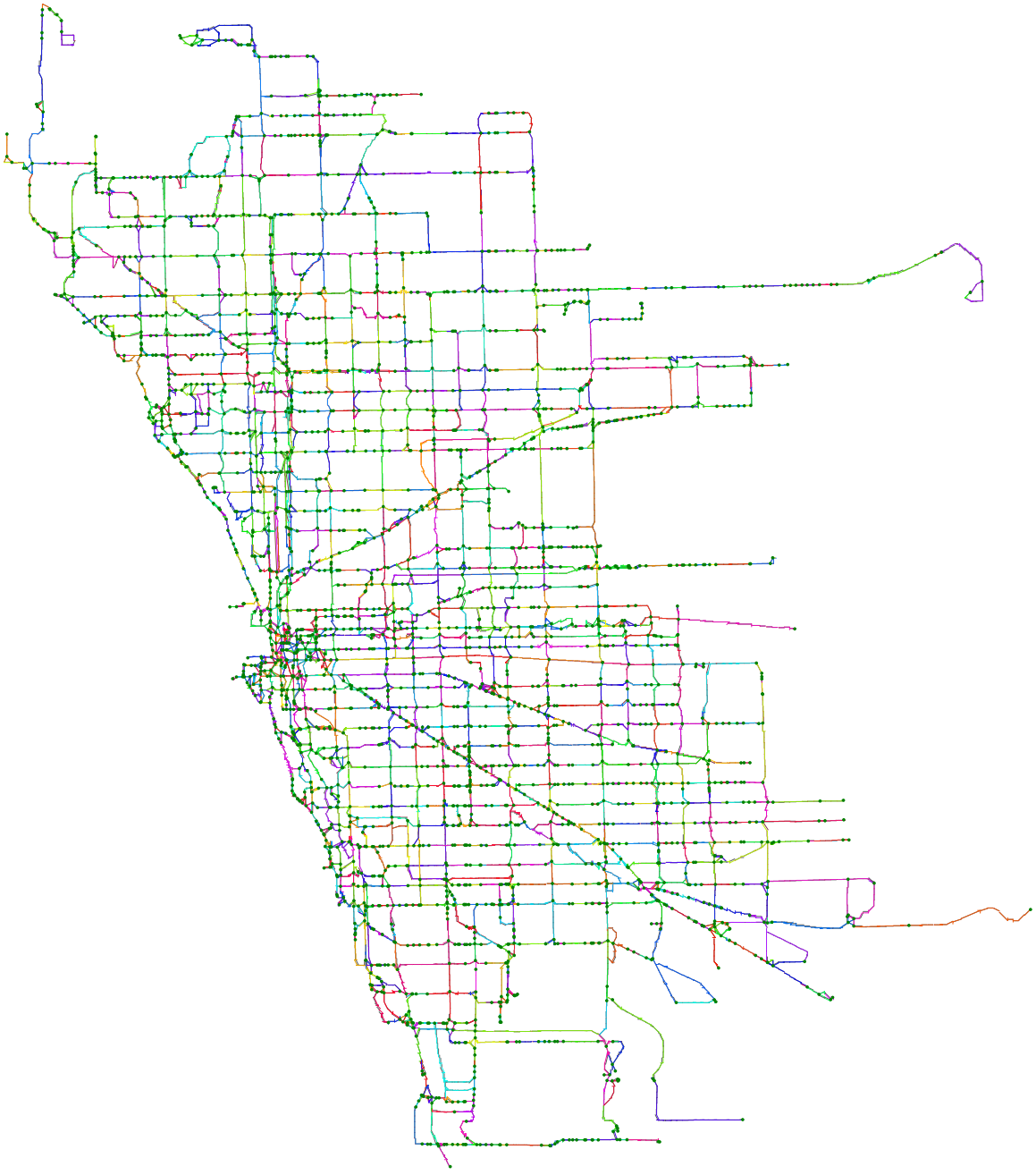


Figure 17: Chicago public transit network network decomposed with the greedy tree decomposition using the same color coding of nodes and edges as in Figure 16.

bundle		$\delta \cdot 10^4$	$\delta_F \cdot 10^4$	δ_F/δ	$n + \ell$	$\lvert S\rvert$	time
Stuttgart	TBD+DP	10.00	9.60	0.96	83 + 102	**36**	1
	BCA	10.00	9.60	0.96	83 + 102	37	5
Chicago	TBD+DP	2.00	1.99	0.99	9,984 + 872	3827	181
	TBD+DP	0.50	0.50	1.00	9,984 + 872	4625	149
	BCA	2.00	3.69	1.84	9,984 + 872	**864**	634
	BCA	1.00	1.92	1.92	9,984 + 872	**1781**	789
	BCA	0.50	0.85	1.60	9,984 + 872	**5023**	456
Path bundle	TBD+DP	2.00	1.96	0.98	10,633 + 17,662	**2681**	472
	BCA	2.00	2.33	1.17	10,633 + 17,662	4915	13,700
	BCA	1.00	1.44	1.44	10,633 + 17,662	5017	12,600

Table 3: Comparison of BCA and TBD+DP on public transit networks and a large path bundle extracted from a road network. Labels are the same as in Table 1 and ℓ is the number of polylines.

within a second even for road network bundles with around 10,000 nodes while BCA takes 30 times longer. In terms of quality, TBD+DP produce comparable or even better results than BCA on the Stuttgart and Freiburg network, and clearly superior results on road network bundles. The instances on which TBD+DP was outperformed by BCA in terms of simplification size are bundles with large grid-like structures as the Chicago and the Manhattan public transit network. Here, our tree decomposition results in a huge set of trees of which we need to keep all root and leaf nodes in the simplification. But especially for large instances, the simplicity and the fast computation time of TBD+DP is a great advantage over BCA; in particular as the TBD is independent of δ and the computations of the individual tree simplifications can easily be parallelized for further improvement.

In Table 3, some results for two of the transit networks as well as a large road network path bundle for TBD+DP as well as BCA are summarized. We observe that TBD+DP is significantly better than BCA on the road network instance in terms of both, quality and running time. But on the Chicago public transit networks, BCA produces the smaller simplification, even if forced to obtain the original δ-constraint (compare the 4601 nodes from TBD+DP to the 1781 from BCA where both solutions obey $\delta \leq 2$). The reason for this is the grid-like structure of that network as shown in Figure 17. But on other instances, TBD+DP produces high-quality simplification results quickly, as illustrated for a large road network bundle in Figure 18.

8 Conclusion and Open Problems

We have generalized the well-known problem of polyline simplification from a single polyline to multiple polylines sharing vertices and line segments, which we have called a polyline bundle. Although efficient algorithms for a single polyline have been known for a long time, we could show that simplifying two or more polylines consistently is a problem that is NP-hard to approximate within a factor of $n^{1/3-\varepsilon}$ for any $\varepsilon > 0$ where n is the number of

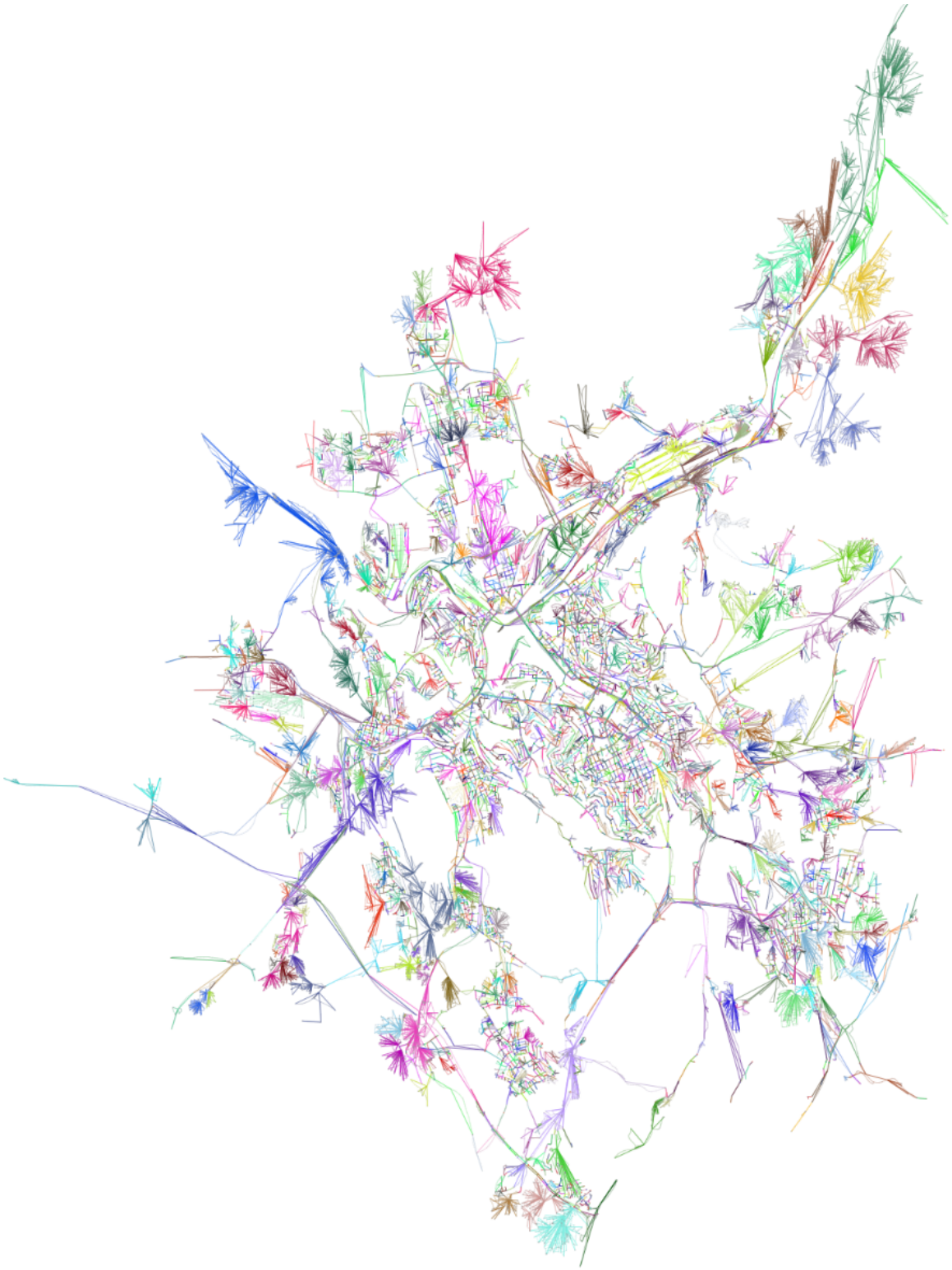
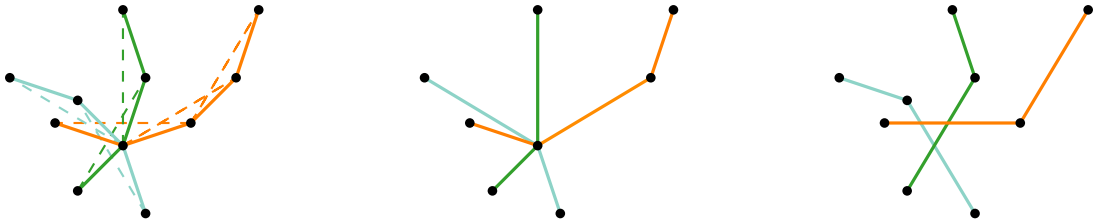


Figure 18: Large polyline bundle extracted from an underlying road network of Stuttgart, together with the TBD+DP based simplification. (North is on the left side.)



(a) initial bundle with shortcuts

(b) simplification with minimum number of vertices

(c) simplification with minimum number of line segments

Figure 19: Optimal simplifications of a polyline bundle (individual polylines are indicated by color) using different objective functions.

vertices in the polyline bundle.

However, if we relax the constraint on the Fréchet distance between original and simplified polyline by a factor of 2, we can overcome this strong inapproximability bound. Moreover, we have seen that we can find an optimal simplification efficiently if we have only a small number of shared vertices (by an FPT algorithm) or if the input instance is a (rooted) tree bundle (by a DP algorithm). If we do not have tree bundles, we suggest some heuristics to decompose a polyline bundle to a set of tree bundles.

Though most of our results are of theoretical nature, we have concluded our work with a small experimental evaluation, which suggest that both our bi-criteria approximation algorithm as well as the dynamic programming approach combined with a tree decomposition heuristic are applicable on real-world instances. The latter performed best on networks with some hierarchical structure like street networks or historically-grown German cities.

Based on our results, there are many possible directions for future research. Specifically, we propose the following theoretical and applied open problems.

- Improve our inapproximability bound of $n^{1/3}$ further or show its tightness.
- Our current bi-criteria approximation guarantee is logarithmic in the number of polylines ℓ plus the number of vertices n. In most practical application, ℓ is smaller than n or at most polynomial in n. From a theoretical perspective, however, it might be interesting to get rid of the dependency on ℓ in the bi-criteria approximation in order to get improvements for the case where ℓ is significantly larger than n. Also investigate whether our bi-criteria approximation factors are tight or can be improved or traded.
- Our result that tree bundles can be processed in polynomial time might hint at parameterizability by, e.g., the treewidth of the union graph of the polylines. So investigating more suitable parameters for an FPT algorithm might be a sensible avenue for future work.
- As a distance measure, we have employed the local Fréchet distance, which we consider to be more natural and intuitive than the local Hausdorff distance when comparing

polylines. However, the local Hausdorff distance is sometimes used in classical polyline simplification as well. Our hardness, FPT, and DP result also apply to the Hausdorff distance, but our bi-criteria approximation algorithm fails since Lemma 7 is not applicable for the Hausdorff distance. Consider approximating PBS using the local Hausdorff distance or other (even non-local) distance measures.

- In our generalization from a single polyline to a polyline bundle, the objective is to minimize the number of retained vertices. However, minimizing the number of retained line segments is an alternative objective function, which also generalizes the classical minimization problem for a single polyline. Optimal simplifications for both objectives may differ; see Figure 19. Our hardness, FPT, and DP results also hold when minimizing the number of retained line segments. However, it is not clear how to obtain a similar result for the bi-criteria approximation algorithm.

- A severe restriction in our definition for simplifying polyline bundles is that the endpoints of each individual polyline need to be kept. For many instances, this reduces the number of possible simplifications significantly. Develop a model that allows removing endpoints of polylines.

- In Figure 1, a shared segment becomes a single vertex in the simplification, which may give a misleading picture. One could require that a sequence of shared segments must not become a single vertex when simplified.

- Also consider non-simple polylines, i.e., polylines were the same vertex can appear multiple times.

- Use the model of polyline bundles in more scenarios occurring in science and beyond.

- Develop new heuristics and practically applicable algorithm for PBS.

- Our experimental studies provided some preliminary insights. New and extended experiments could be conducted on more real-world data also incorporating new heuristic, approximation, or exact algorithms for comparison.

References

[1] P. K. Agarwal, S. Har-Peled, N. H. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005. doi:10.1007/s00453-005-1165-y.

[2] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk. *Map Construction Algorithms*. Springer, 2015. doi:10.1007/978-3-319-25166-0.

[3] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

[4] S. Bereg, M. Jiang, W. Wang, B. Yang, and B. Zhu. Simplifying 3D polygonal chains under the discrete Fréchet distance. In *Proc. 8th Latin American Symposium on Theoretical Informatics (LATIN'08)*, pages 630–641. Springer, 2008. doi:10.1007/978-3-540-78773-0_54.

[5] M. Brankovic, K. Buchin, K. Klaren, A. Nusser, A. Popov, and S. Wong. (k, l)-medians clustering of trajectories using continuous dynamic time warping. In *Proc. 28th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 99–110, 2020. doi:10.1145/3397536.3422245.

[6] K. Bringmann and B. R. Chaudhury. Polyline simplification has cubic complexity. *Journal of Computational Geometry*, 11(2):94–130, 2021. doi:10.20382/jocg.v11i2a5.

[7] K. Buchin, A. Driemel, J. Gudmundsson, M. Horton, I. Kostitsyna, M. Löffler, and M. Struijs. Approximating (*k, l*)-center clustering for curves. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*, pages 2922–2938. SIAM, 2019. doi:10.1137/1.9781611975482.181.

[8] K. Buchin, A. Driemel, and M. Struijs. On the hardness of computing an average curve. In *Proc. 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'20)*, pages 19:1–19:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SWAT.2020.19.

[9] M. Buchin, A. Driemel, and D. Rohde. Approximating (*k, l*)-median clustering for polygonal curves. *ACM Trans. Algorithms*, 19(1):4:1–4:32, 2023. doi:10.1145/3559764.

[10] M. Buchin, A. Driemel, and B. Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. In *Proc. 30th Annual Symposium on Computational Geometry (SoCG'14)*, pages 367–376, 2014. doi:10.1145/2582112.2582144.

[11] M. Buchin, B. Kilgus, and A. Kölzsch. Group diagrams for representing trajectories. *International Journal of Geographical Information Science*, 34(12):2401–2433, 2020. doi:10.1080/13658816.2019.1684498.

[12] M. Buchin, I. van der Hoog, T. Ophelders, L. Schlipf, R. I. Silveira, and F. Staals. Efficient Fréchet distance queries for segments. In *Proc. 30th Annual European Symposium on Algorithms (ESA'22)*, pages 29:1–29:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.29.

[13] 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. doi:10.1142/S0218195996000058.

[14] S. Cheng and H. Huang. Curve simplification and clustering under Fréchet distance. In *Proc. 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'23)*, pages 1414–1432. SIAM, 2023. doi:10.1137/1.9781611977554.ch51.

[15] M. de Berg, M. J. van Kreveld, and S. Schirra. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Systems*, 25(1):243–257, 1998. doi:10.1559/152304098782383007.

[16] 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. doi:10.3138/fm57-6770-u75u-7727.

[17] A. Driemel, A. Krivošija, and C. Sohler. Clustering time series under the Fréchet distance. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'16)*, pages 766–785. SIAM, 2016. doi:10.1137/1.9781611974331.ch55.

[18] R. Estkowski and J. S. B. Mitchell. Simplifying a polygonal subdivision while keeping it simple. In *Proc. 17th Annual Symposium on Computational Geometry (SoCG'01)*, pages 40–49. ACM, 2001. doi:10.1145/378583.378612.

[19] C. Fan, O. Filtser, M. J. Katz, T. Wylie, and B. Zhu. On the chain pair simplification problem. In *Proc. 14th International Symposium on Algorithms and Data Structures (WADS'15)*, pages 351–362. Springer, 2015. doi:10.1007/978-3-319-21840-3_29.

[20] C. Fan, O. Filtser, M. J. Katz, and B. Zhu. On the general chain pair simplification problem. In *Proc. 41st International Symposium on Mathematical Foundations of Computer Science (MFCS'16)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.37.

[21] S. Funke, T. Mendel, A. Miller, S. Storandt, and M. Wiebe. Map simplification with topology constraints: Exactly and in practice. In *Proc. 19th Workshop on Algorithm Engineering and Experiments (ALENEX'17)*, pages 185–196. SIAM, 2017. doi:10.1137/1.9781611974768.15.

[22] M. Godau. A natural metric for curves – Computing the distance for polygonal chains and approximation algorithms. In *Proc. 8th Annual Symposium on Theoretical Aspects of Computer Science (STACS'91)*, pages 127–136, 1991. doi:10.1007/BFb0020793.

[23] L. J. Guibas, J. Hershberger, J. S. B. Mitchell, and J. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *International Journal of Computational Geometry and Applications*, 3(4):383–415, 1993. doi:10.1142/S0218195993000257.

[24] M. M. Halldórsson. Approximating the minimum maximal independence number. *Information Processing Letters*, 46(4):169–172, 1993. doi:10.1016/0020-0190(93)90022-2.

[25] S. He, F. Bastani, S. Abbar, M. Alizadeh, H. Balakrishnan, S. Chawla, and S. Madden. RoadRunner: improving the precision of road network inference from GPS trajectories. In *Proc. 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 3–12, 2018. doi:10.1145/3274895.3274974.

[26] J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line-simplification algorithm. In *Proc. 5th International Symposium on Spatial Data Handling (SDH'92)*, pages 134–143, 1992.

[27] H. Imai and M. Iri. Polygonal approximations of a curve – formulations and algorithms. In G. T. Toussaint, editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 71–86. North-Holland, 1988. doi:10.1016/B978-0-444-70467-2.50011-4.

[28] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974. doi:10.1016/S0022-0000(74)80044-9.

[29] A. Melkman and J. O'Rourke. On polygonal chain approximation. In G. T. Toussaint, editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 87–95. North-Holland, 1988. doi:10.1016/B978-0-444-70467-2.50012-6.

[30] N. Meratnia and R. A. de By. Spatiotemporal compression techniques for moving point objects. In *Proc. 9th International Conference on Extending Database Technology (EDBT'04)*, pages 765–782. Springer, 2004. doi:10.1007/978-3-540-24741-8_44.

[31] MobilityData IO. OpenMobilityData. URL: https://transitfeeds.com.

[32] OpenStreetMap contributors. Planet dump retrieved from planet.osm.org, 2017. URL: https://www.openstreetmap.org.

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

[34] P. Schäfer, S. Storandt, and J. Zink. Optimal polyline simplification under the local Fréchet distance in (near-)quadratic time. In *Proc. 35th Canadian Conference on Computational Geometry (CCCG'23)*, pages 225–238, 2023. URL: https://wadscccg2023.encs.concordia.ca/assets/pdf/CCCG_2023_proc.pdf#section.0.26, arXiv:2201.01344.

[35] M. van de Kerkhof, I. Kostitsyna, M. Löffler, M. Mirzanezhad, and C. Wenk. Global curve simplification. In *Proc. 27th Annual European Symposium on Algorithms (ESA'19)*, pages 67:1–67:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.67.

[36] M. J. van Kreveld, M. Löffler, and L. Wiratma. On optimal polyline simplification using the Hausdorff and Fréchet distance. *Journal of Computational Geometry*, 11(1):1–25, 2020. doi:10.20382/jocg.v11i1a1.

[37] S. Wu and M. R. G. Márquez. A non-self-intersection Douglas-Peucker algorithm. In *Proc. 16th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'03)*, pages 60–66. IEEE Computer Society, 2003. doi:10.1109/sibgra.2003.1240992.

[38] T. Wylie and B. Zhu. Protein chain pair simplification under the discrete Fréchet distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(6):1372–1383, 2013. doi:10.1109/tcbb.2013.17.