Recognizing Weakly Simple Polygons*

Hugo A. Akitaya† Greg Aloupis† Jeff Erickson‡ Csaba D. Tóth†§

Abstract

We present an $O(n \log n)$-time algorithm that determines whether a given n-gon in the plane is weakly simple. This improves upon an $O(n^2 \log n)$-time algorithm by Chang, Erickson, and Xu [6]. Weakly simple polygons are required as input for several geometric algorithms. As such, recognizing simple or weakly simple polygons is a fundamental problem.

Keywords: simple polygon, combinatorial embedding, perturbation

MSC: 05C10, 05C38, 52C45, 68R10.

1 Introduction

A polygon is *simple* if it has distinct vertices and interior-disjoint edges that do not pass through vertices. Geometric algorithms are often designed for simple polygons, but many also work for degenerate polygons that do not "self-cross." A polygon with at least three vertices is *weakly simple* if for every $\varepsilon > 0$, the vertices can be perturbed within a ball of radius ε to obtain a simple polygon. Such polygons arise naturally in numerous applications, e.g., for modeling planar networks or as the geodesic hull of points within a simple polygon (Figure 1).

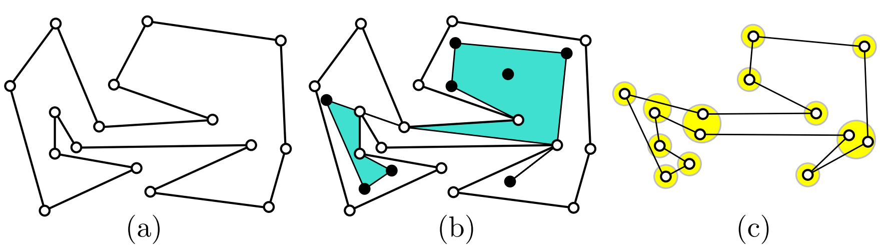


Figure 1: (a) A simple polygon P with 16 vertices. (b) Eight points in the interior of P (solid dots); their geodesic hull is a weakly simple polygon P' with 14 vertices. (c) A perturbation of P' into a simple polygon.

*A preliminary version of this paper appeared in the *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG 2016)*, doi:10.4230/LIPIcs.SoCG.2016.8.

†Department of Computer Science, Tufts University, Medford, MA. Email: hugo.alves_akitaya@tufts.edu, aloupis.greg@gmail.com, cdtoth@eecs.tufts.edu

‡Department of Computer Science, University of Illinois, Urbana-Champaign, IL. Email: jeffe@illinois.edu

§Department of Mathematics, California State University Northridge, Los Angeles, CA.

Several alternative definitions have been proposed for weakly simple polygons, formalizing the intuition that such polygons do not self-cross. Some of these definitions were unnecessarily restrictive or incorrect; see [6] for a detailed discussion and five equivalent definitions for weak simplicity of a polygon. Among others, a result by Ribó Mor [16, Theorem 3.1] implies an equivalent definition in terms of Fréchet distance, in which a polygon is perturbed into a simple closed curve (see Section 2). This definition is particularly useful for recognizing weakly simple polygons, since it allows transforming edges into polylines (by subdividing the edges with Steiner points, which may be perturbed). With suitable Steiner points, the perturbation of a vertex incurs only local changes. (In other words, we do not need to worry about stretchability of the perturbed configuration.)

We can decide whether an n-gon in the plane is simple in $O(n \log n)$ time by a sweepline algorithm [17]. Chazelle's polygon triangulation algorithm also recognizes simple polygons (in $O(n)$ time), because it only produces a triangulation if the input is simple [7]. Recognizing weakly simple polygons, however, is more subtle. Skopenkov [18] gave a combinatorial characterization of the topological obstructions to weak simplicity in terms of line graphs. Cortese et al. [10] gave an $O(n^6)$-time algorithm to recognize weakly simple n-gons. Chang et al. [6] improved the running time to $O(n^2 \log n)$ in general; and to $O(n \log n)$ in several special cases. They identified two features that are difficult to handle: A *spur* is a vertex whose incident edges overlap, and a *fork* is a vertex that lies in the interior of an edge. (A vertex may be both a fork and a spur.) They gave an easy algorithm for polygons that have neither forks nor spurs, and two more involved ones for polygons with spurs but no forks and for polygons with forks but no spurs, all three running in $O(n \log n)$ time. In the presence of both forks and spurs, they presented an $O(n^2 \log n)$ time algorithm that eliminates forks by subdividing all edges that contain vertices in their interiors, potentially creating a quadratic number of vertices.

We show how to manage both forks and spurs efficiently, while building on ideas from [6, 10] and from Arkin et al. [2], and obtain the following main results.

Theorem 1. *Deciding whether a polygon P with n vertices in the plane is weakly simple takes $O(n \log n)$ time.*

Theorem 2. *Given a weakly simple polygon P with n vertices and a constant $\varepsilon > 0$, a simple polygon with $2n$ vertices within Fréchet distance ε from P can be computed in $O(n \log n)$ time.*

Our decision algorithm is detailed in Sections 3–5. It consists of three phases, simplifying the input polygon by a sequence of reduction steps. First, the *preprocessing* phase rules out edge crossings in $O(n \log n)$ time and applies known reduction steps such as *crimp reductions* and *node expansions* (Section 3). Second, the *bar simplification* phase successively eliminates all forks (Section 4). Third, the *spur elimination* phase eliminates all spurs (Section 5). When neither forks nor spurs are present, we can decide weak simplicity in $O(n)$ time [10]. Finally, by reversing the sequence of operations, we can also perturb any weakly simple polygon into a simple polygon in $O(n \log n)$ time (Section 6).

2 Preliminaries

In this section, we review previously established definitions and known methods from [6] and [10].

Polygons and weak simplicity. An *arc* in $\mathbb{R}^2$ is a continuous function $\gamma : [0,1] \to \mathbb{R}^2$. A *closed curve* is a continuous function (map) $\gamma : \mathbb{S}^1 \to \mathbb{R}^2$. A closed curve γ is *simple* (also known as a *Jordan curve*) if it is injective. A (*simple*) *polygon* is the image of a piecewise linear (*simple*) closed curve. Thus a polygon P can be represented by a cyclic sequence of points $(p_0, \ldots, p_{n-1})$, called *vertices*, where the image of γ consists of line segments $p_0p_1, \ldots, p_{n-2}p_{n-1}$, and $p_{n-1}p_0$ in this cyclic order. Note that a nonsimple polygon may have repeated vertices and overlapping edges [14]. Similarly, a *polygonal chain* (alternatively, *path*) is the image of a piecewise linear arc, and can be represented by a sequence of points $[p_0, \ldots, p_{n-1}]$.

A polygon $P = (p_0, \ldots, p_{n-1})$ is *weakly simple* if $n = 2$, or if $n > 2$ and for every $\varepsilon > 0$ there is a simple polygon $(p'_0, \ldots, p'_{n-1})$ such that $|p_i, p'_i| < \varepsilon$ for all $i = 0, \ldots, n-1$. This definition is difficult to work with because a small perturbation of a vertex modifies the two incident edges, which may be long, and the effect of a perturbation is not localized. Combining earlier results from [9], [10], and [16, Theorem 3.1], an equivalent definition was formulated by Chang et al. [6] in terms of Fréchet distance: A polygon given by $\gamma : \mathbb{S}^1 \to \mathbb{R}^2$ is weakly simple if for every $\varepsilon > 0$ there is a simple closed curve $\gamma' : \mathbb{S}^1 \to \mathbb{R}^2$ such that $\mathrm{dist}_F(\gamma, \gamma') < \varepsilon$, where dist_F denotes the Fréchet distance between two closed curves. The curve γ' can approximate an edge of the polygon by a polyline, and any perturbation of a vertex can be restricted to a small neighborhood. With this definition, recognizing weakly simple polygons becomes a combinatorial problem, as explained below. Note that in topology, the broader question of *isotopic embeddability* has been considered [15, 18]: Given a continuous map $f : A \to \mathbb{R}^d$ for a simplicial complex A, is it isotopic to some *injective* continuous map (i.e., *embedding*) $g : A \to \mathbb{R}^d$?

Bar decomposition and image graph. Two edges of a polygon P *cross* if their interiors intersect at precisely one point; we call this an *edge crossing*. Weakly simple polygons cannot have edge crossings. In the remainder of this section, we assume that such crossings have been ruled out. Two edges of P *overlap* if their intersection is a (nondegenerate) line segment. The transitive closure of the overlap relation is an equivalence relation on the edges of P; see Figure 2(a) where equivalence classes are represented by purple regions. The union of all edges in an equivalence class is called a *bar*.[1] All bars of a polygon can be computed in $O(n \log n)$ time [6]. The bars are open line segments that are pairwise disjoint. There are at most n bars, since the bars are unions of disjoint subsets of edges.

The vertices and bars of P define a planar straight-line graph G, called the *image graph* of P. We call the vertices and edges of G *nodes* and *segments*[1] to distinguish them from the vertices and edges of P. Every node that is not in the interior of a bar is called *sober*[1]. The set of nodes in G is $\{p_0, \ldots, p_{n-1}\}$ (note that P may have repeated vertices that correspond to the same node); two nodes are connected by a segment in G if they are consecutive nodes along a bar; see Figure 2(b). Hence G has $O(n)$ nodes and segments, and it can be computed in $O(n \log n)$ time [6]. Note, however, that up to $O(n)$ edges of P may pass through a node of G, and there may be $O(n^2)$ edge-node pairs such that an edge of P passes through a node of G. An $O(n \log n)$-time algorithm cannot afford to compute these pairs explicitly.

Operations. We use certain elementary operations that successively modify a polygon and ultimately eliminate forks and spurs. An operation that produces a weakly simple polygon if and only if it is performed on a weakly simple polygon is called *ws-equivalent*. Several such operations are already known (e.g., crimp reduction, node expansion, bar expansion). We shall use these and introduce several new operations in Sections 3.3–5.

Combinatorial characterization of weak simplicity. To show that an operation is ws-

[1] We adopt terminology from [6].

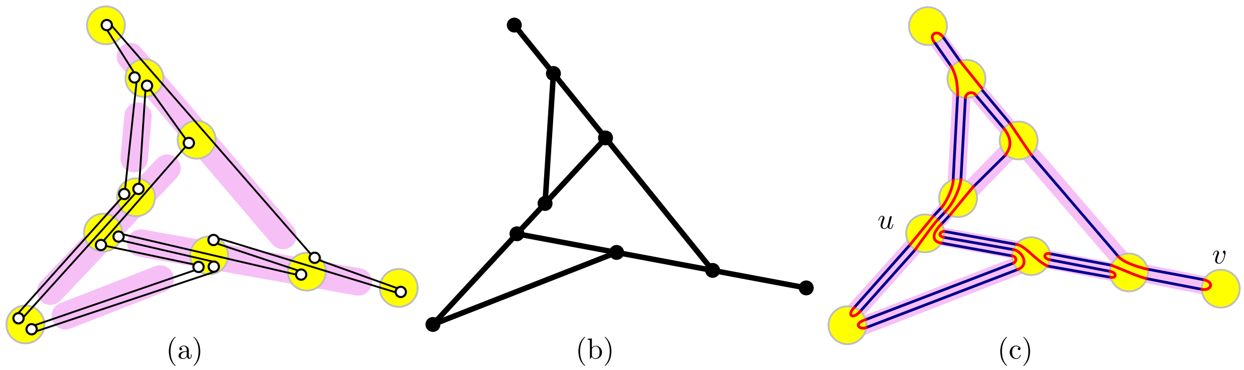


Figure 2: (a) The bar decomposition for a weakly simple polygon P with 16 vertices (P is perturbed into a simple polygon for clarity). (b) The image graph of P. (c) A perturbation in a strip system of P.

equivalent, it suffices to provide suitable simple ε-perturbations for all $\varepsilon > 0$. We use a combinatorial representation of an ε-perturbation (independent of ε or any specific embedding). When a weakly simple polygon P is perturbed into a simple polygon, overlapping edges in P are perturbed into interior-disjoint near-parallel edges, which define an ordering. It turns out that these orderings over all segments of the image graph are sufficient to encode an ε-perturbation and to (re)construct an ε-perturbation.

We rely on the notion of "strip system" introduced in [6, Appendix B]. Similar concepts have previously been used in [9, 10, 11, 15, 18]. Let P be a polygon and G its image graph. Without loss of generality, we assume that no bar is vertical (so that the above-below relationship is defined between disjoint segments parallel to a bar). For every $\varepsilon > 0$, the *ε-strip-system* of P consists of the following regions:

- For every node u of G, let D_u be a disk of radius ε centered at u.
- For every segment uv, let the *corridor* N_{uv} be the set of points at distance at most ε^2 from uv, outside of the disks D_u and D_v, that is, $N_{uv} = \{p \in \mathbb{R}^2 : \text{dist}(p, uv) \leq \varepsilon^2, p \notin D_u \cup D_v\}$.

Denote by U_ε the union of all these disks and corridors. There is a sufficiently small $\varepsilon_0 = \varepsilon_0(P) > 0$, depending on P, such that the disks D_u are pairwise disjoint, the corridors N_{uv} are pairwise disjoint, and every corridor N_{uv} of a segment intersects only the disks at its endpoints D_u and D_v. These properties hold for all ε, $0 < \varepsilon < \varepsilon_0$.

A polygon is *in the ε-strip-system* of P if its edges alternate between an edge that connects the boundaries of two disks D_u and D_v and whose interior is contained in N_{uv}; and an edge between two points on the boundary of a disk. In particular, the edges of P that lie in a disk D_u or a corridor N_{uv} form a perfect matching. See Figure 2(c) for an example, where the edges within the disk D_u are drawn with circular arcs for clarity. Let $\Phi(P)$ be the set of simple polygons in the ε-strip-system of P that cross the disks and corridors in the same order as P traverses the corresponding nodes and segments of G. It is clear that every $Q \in \Phi(P)$ is within Fréchet distance ε from P. By [6, Theorem B.2], P is weakly simple if and only if $\Phi(P) \neq \emptyset$.

Combinatorial representation by signatures. Let Q be a polygon in the strip system of P. For each segment uv, the above-below relationship of the edges of Q in N_{uv} is a total order. We define the *signature* of $Q \in \Phi(P)$, denoted $\sigma(Q)$, as the collection of these total orders for all segments of G.

Given the signature $\sigma(Q)$ of a polygon Q in the strip system of P, we can easily (re)construct a simple polygon Q' with the same signature in the ε-strip-system of P for any $0 < \varepsilon < \varepsilon_0$. For every segment uv of G, let the *volume* $\mathrm{vol}(uv)$ be the number of edges of P that lie on uv. Place $\mathrm{vol}(uv)$ parallel line segments between ∂D_u and ∂D_v in N_{uv} of the ε-strip-system of P. Finally, for every disk D_u, construct a straight-line perfect matching between the endpoints of these edges that lie in ∂D_u: connect the endpoints of two edges if they correspond to adjacent edges of P. It is easily verified that the Fréchet distance between Q and Q' is at most 2ε. Furthermore, $Q \in \Phi(P)$ implies $Q' \in \Phi(P)$, since Q and Q' determine the same perfect matching between corresponding endpoints on ∂D_u at every node u.

Remark 1. The construction above has two consequences: (1) To prove weak simplicity, it is enough to find a signature that defines a simple perturbation. In other words, the signature can witness weak simplicity (independent of the value of ε). (2) Weak simplicity of a polygon depends only on the *combinatorial embedding* of the image graph G (i.e., the counterclockwise order of edges incident to each vertex), as long as G is a planar graph. Consequently, when an operation modifies the image graph, it is enough to maintain the combinatorial embedding of G (the precise coordinates of the nodes do not matter).

In the presence of spurs, the size of a signature is $O(n^2)$, and this bound is the best possible. We use this simple combinatorial representation in our proofs of correctness, but our algorithm does not maintain it explicitly. In Section 6, we introduce another combinatorial representation of $O(n)$ size that uses the ordering of the edges in each bar (rather than each segment) of the image graph.

Combinatorially different perturbations. In the absence of spurs, a polygon P determines a unique noncrossing perfect matching in each disk D_u, hence a unique noncrossing 2-regular graph in the ε-strip-system of P [6, Section 3.3]. Consequently, to decide whether P is weakly simple it is enough to check whether this graph is connected. The uniqueness no longer holds in the presence of spurs. In fact, it is not difficult to construct weakly simple n-gons that admit $2^{\Theta(n)}$ perturbations into simple polygons that are combinatorially different (i.e., have different bar-signatures); see Figure 3.

Figure 3: Two perturbations of a weakly simple polygon on 6 vertices (all of them spurs) that alternate between two distinct points in the plane.

3 Preprocessing

We are given a polygon $P = (p_0, \ldots, p_{n-1})$ in the plane. By a standard line sweep [17], we can test whether any two edges properly cross; if they do, the algorithm halts and reports that P is not weakly simple. We then simplify the polygon, using some known steps from [2, 6], and some new ones. All of this takes $O(n \log n)$ time.

3.1 Crimp reduction

Arkin et al. [2] gave an $O(n)$-time algorithm for recognizing weakly simple n-gons in the special case where all edges are collinear (in the context of flat foldability of a polygonal linkage). They defined the ws-equivalent crimp-reduction operation. A *crimp* is a chain of three consecutive collinear edges $[a, b, c, d]$ such that both the first edge $[a, b]$ and the last edge $[c, d]$ contain the middle edge $[b, c]$ (the containment need not be proper). The operation crimp-reduction(a, b, c, d) replaces the crimp $[a, b, c, d]$ with edge $[a, d]$; see Figure 4.

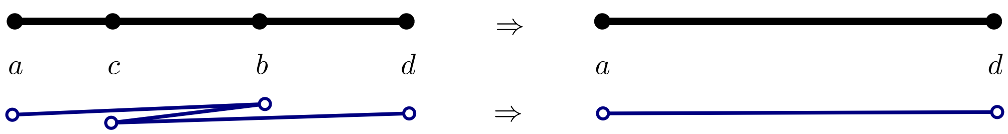


Figure 4: A crimp reduction replaces $[a, b, c, d]$ with $[a, d]$. Top: image graph. Bottom: polygon.

Lemma 1. *The* crimp-reduction *operation is ws-equivalent.*

Proof. Let P_1 and P_2 be two polygons such that P_2 is obtained from P_1 by the operation crimp-reduction(a, b, c, d). Without loss of generality, assume that ad is horizontal with a on the left and d on the right.

First assume that P_1 is weakly simple. Then there exists a simple polygon $Q_1 \in \Phi(P_1)$. We modify Q_1 to obtain a simple polygon $Q_2 \in \Phi(P_2)$. Without loss of generality, assume that edge $[a, b]$ is above $[b, c]$ (consequently, $[c, d]$ is below $[b, c]$) in Q_1. The modification involves the perfect matchings at the disks D_b and D_c, and all disks and corridors along the line segment bc. Denote by W_{top} the set of maximal paths that lie in the convex hull of $D_b \cup D_c$, below $[a, b]$ and above $[b, c]$; similarly, let W_{bot} be the set of maximal paths that lie in the convex hull of $D_b \cup D_c$, below $[b, c]$ and above $[c, d]$. We proceed in two steps; refer to Figure 5. First, replace the path $[a, b, c, d]$ with the path $[a, c, b, d]$ such that the new edge $[a, c]$ replaces the old $[a, b]$ in the edge ordering of segment ac, the new $[c, b]$ replaces $[b, c]$ in the segments contained in bc, and finally the new $[b, d]$ replaces $[c, d]$ in bd. Second, exchange W_{top} and W_{bot} such that the top-to-bottom order within each set of paths remains the same. Since the top-to-bottom order within W_{top} and W_{bot} is preserved, and the paths in W_{top} (resp., W_{bot}) lie below (resp., above) the new path $[a, c, b, d]$, no edge crossings have been introduced. We obtain a simple polygon $Q_2 \in \Phi(P_2)$, which shows that P_2 is weakly simple.

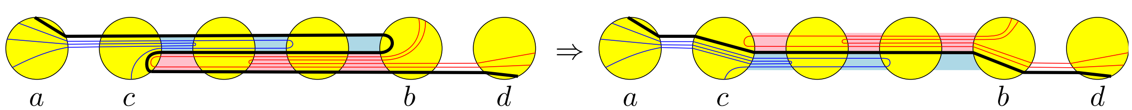


Figure 5: The operation crimp-reduction replaces a crimp $[a, b, c, d]$ with an edge $[ad]$.

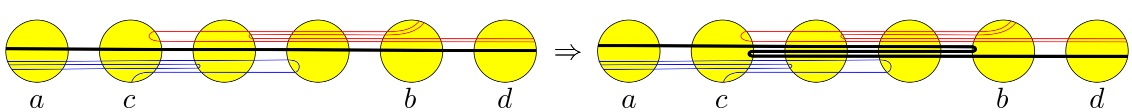


Figure 6: The reversal of crimp-reduction replaces edge $[ad]$ with a crimp $[a, b, c, d]$.

Next assume that P_2 is weakly simple. Then, there exists a simple polygon $Q_2 \in \Phi(P_2)$. We modify Q_2 to obtain a simple polygon $Q_1 \in \Phi(P_1)$; refer to Figure 6. Replace edge $[a, d]$ by $[a, b, c, d]$

also replacing $[a, d]$ in the ordering of the affected segments by $[c, d]$, $[b, c]$, and $[a, b]$, in this order. The new ordering produces a polygon Q_1 in the strip system of P. Because Q_2 is simple, by construction the new matchings do not interact with the preexisting edges in the disks. Hence, $Q_1 \in \Phi(P_1)$, which shows that P_1 is weakly simple. ☐

Given a chain of two edges $[a, b, c]$ such that $[a, b]$ and $[b, c]$ are collinear but do not overlap, the merge operation replaces $[a, b, c]$ with a single edge $[a, c]$. The merge operation (as well as its inverse, subdivision) is ws-equivalent by the definition of weak simplicity in terms of Fréchet distance [6]. If we greedily apply crimp-reduction and merge operations, in linear time we obtain a polygon with the following two properties:

(A1) Every two consecutive collinear edges overlap (i.e., form a spur).

(A2) No three consecutive collinear edges form a crimp.

Assuming properties (A1) and (A2), we can characterize a chain of collinear edges with the sequence of their edge lengths.

Lemma 2. *Let $C = [e_i, \ldots, e_k]$ be a chain of collinear edges in a polygon with properties (A1) and (A2). Then the sequence of edge lengths $(|e_i|, \ldots, |e_k|)$ is unimodal (all local maxima are consecutive); and no two consecutive edges have the same length, except possibly the maximal edge length that can occur at most twice.*

Proof. For every j such that $i < j < k$, consider $|e_j|$. If $|e_{j-1}|$ and $|e_{j+1}|$ are at least as large as $|e_j|$, then the three edges form a crimp, by (A1). However, this contradicts (A2). This proves unimodality, and that no three consecutive edges can have the same length. In fact if $|e_j|$ is not maximal, one neighbor must be strictly smaller, to avoid the same contradiction. ☐

The operations introduced in Section 4 maintain properties (A1)–(A2) for all maximal paths inside an elliptical disk D_b.

3.2 Node expansion

Compute the bar decomposition of P and its image graph G (defined in Section 2, see Figure 2). For every sober node of the image graph, we perform the ws-equivalent node-expansion operation, described in [6, Section 3] (Cortese et al. [10] call this a *cluster expansion*). Let u be a sober node of the image graph. Let D_u be the disk centered at u with radius $\delta > 0$ sufficiently small so that D_u intersects only the segments incident to u. For each segment ux incident to u, create a new node u^x at the intersection point $ux \cap \partial D_u$. Then modify P by replacing each subpath $[x, u, y]$ passing through u by $[x, u^x, u^y, y]$; see Figure 7. If a node expansion produces an edge crossing, report that P is not weakly simple.

3.3 Bar expansion

Chang et al. [6, Section 4] define a bar expansion operation. In this paper, we refer to it as old-bar-expansion. For a bar b of the image graph, draw a long and narrow ellipse D_b around the interior nodes of b, create subdivision vertices at the intersection of ∂D_b with the edges, and replace each maximal path in D_b by a straight-line edge. If b contains no spurs, old-bar-expansion is known to

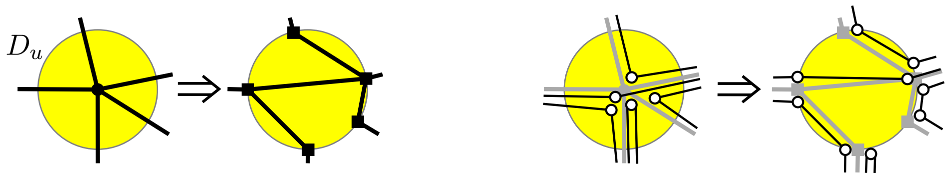


Figure 7: Node expansion. (Left) Changes in the image graph. (Right) Changes in P (the vertices are perturbed for clarity). New nodes are shown as squares.

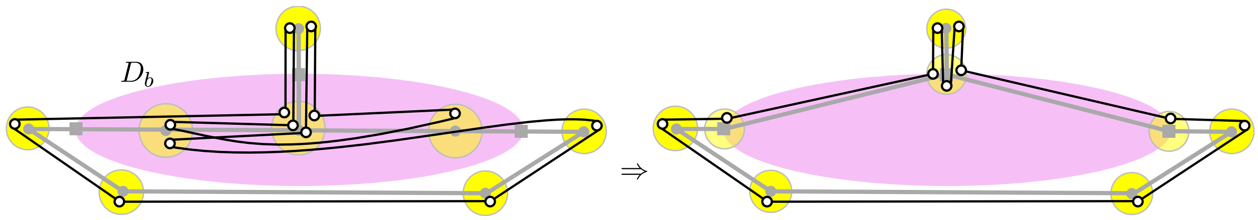


Figure 8: The old-bar-expansion converts a non-weakly simple polygon to a weakly simple one.

be ws-equivalent [6]. Otherwise, it can produce false positives, hence it is not ws-equivalent; see Figure 8 for an example.

New bar expansion operation. Let b be a bar in the image graph with at least one interior node; see Figure 9. Without loss of generality, assume that b is horizontal. Let D_b be an ellipse whose major axis is in b such that D_b contains all interior nodes of b (nodes in b except its endpoints), but does not contain any other node of the image graph and does not intersect any segment that is not incident to some node inside D_b.

Similar to old-bar-expansion, the operation new-bar-expansion introduces subdivision vertices on ∂D_b, however we keep all interior vertices of a bar at their original positions. In Section 4, we apply a sequence of new operations to eliminate all vertices on b sequentially while creating new nodes in the vicinity of D_b. Our bar expansion operation can be considered as a preprocessing step for this subroutine.

For each segment ux between a node $u \in b \cap D_b$ and a node $x \notin b$, create a new node u^x at the intersection point $ux \cap \partial D_b$ and subdivide every edge $[u, x]$ to a path $[u, u^x, x]$. For each endpoint v of b, create two new nodes, v' and v'', as follows. Node v is adjacent to a unique segment $vw \subset b$, where $w \in b \cap D_b$. Create a new node $v' \in \partial D_b$ sufficiently close to the intersection point $vw \cap \partial D_b$, but strictly above b; and create a new node v'' in the interior of segment $vw \cap D_b$. Subdivide every edge $[v, y]$, where $y \in b$, into a path $[v, v', v'', y]$. Since the new-bar-expansion operation consists of only subdivisions (and slight perturbations of the edges passing through the end-segments of the bars), it is ws-equivalent.

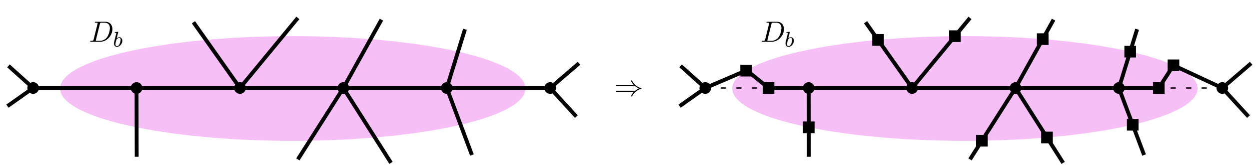


Figure 9: The changes in the image graph caused by new-bar-expansion.

Crossing paths. Apart from node-expansion and old-bar-expansion, none of our operations creates edge crossings. In some cases, our bar simplification algorithm (Section 4) detects whether two subpaths cross. Crossings between overlapping paths are not easy to identify (see [6, Section 2] for a discussion). We rely on the following simple condition to detect some (but not all) crossings.

Lemma 3. *Let P be a weakly simple polygon parameterized by a curve $\gamma_1 : \mathbb{S}^1 \to \mathbb{R}^2$; and let $\gamma_2 : \mathbb{S}^1 \to \mathbb{R}^2$ be a closed Jordan curve that does not pass through any vertices of P and intersects every edge of P transversely. Suppose that $q_1, \ldots, q_4$ are distinct points in $\gamma_2(\mathbb{S}^1)$ in counterclockwise order. Then there are no two disjoint arcs $I_1, I_2 \subset \mathbb{S}^1$ such that $\gamma_1(I_1)$ and $\gamma_1(I_2)$ connect q_1 to q_3 and q_2 to q_4, each passing through the interior of $\gamma_2(\mathbb{S}^1)$.*

Proof. Suppose, to the contrary, that there exist two disjoint arcs $I_1, I_2 \subset \mathbb{S}_1$ such that $\gamma_1(I_1)$ and $\gamma_1(I_2)$ respectively connect q_1 to q_3 and q_2 to q_4, passing through the interior of $\gamma_2(\mathbb{S}^1)$. (See Figure 10.) Since P is weakly simple, then γ_1 can be perturbed to a closed Jordan curve γ_1' with the same properties as γ_1. Let U denote the interior of $\gamma_2(\mathbb{S}^1)$, and note that U is simply connected. Consequently, $U \setminus \gamma_1'(I_1)$ has two components, which are incident to q_2 and q_4, respectively. The Jordan arc $\gamma_1'(I_2)$ connects q_2 to q_4 via U, so it must intersect $\gamma_1'(I_1)$, contradicting the assumption that γ_1' is a Jordan curve. □

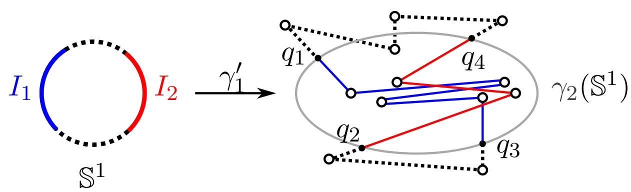


Figure 10: Forbidden configuration described by Lemma 3.

We show that a weakly simple polygon cannot contain certain configurations, outlined below.

Corollary 1. *A weakly simple polygon cannot contain a pair of paths of the following types:*

1. *$[u_1, u_2, u_3]$ and $[v, u_2, w]$, where u_2u_1, u_2v, u_2u_3, and u_2w are nonoverlapping segments in this cyclic order around u_2 (*node crossing*; see Figure 11(a)).*
2. *$[u_1, u_3, w]$ and $[v, u_2, u_4]$, where u_1, u_2, u_3, and u_4 are on a line in this order, and nodes v and w lie in an open halfplane bounded by this line (Figure 11(b)).*
3. *$[u_1, u_2, u_3]$ and $[v_1, v_2, \ldots, v_{k-1}, v_k]$ where $v_2 \in \mathrm{int}(u_2u_3)$, $v_3, \ldots, v_{k-1} \in \{u_2\} \cup \mathrm{int}(u_2u_3)$, nodes u_1 and v_1 lie in an open halfplane bounded by the supporting line of u_2u_3, and node v_k lies on the other open halfplane bounded by this line (Figure 11(c)).*

Proof. In all four cases, Lemma 3 with a suitable Jordan curve γ_2 completes the proof. In case 1, let γ_2 be a small circle around u_2. In case 2, let γ_2 be a small neighborhood of segment u_1u_2. In case 3, let γ_2 be a small neighborhood of the convex hull of $\{v_2, \ldots, v_{k-1}\}$. □

Terminology. We classify the maximal paths in D_b. All nodes $u \in \partial D_b$ lie either above or below b. We call them *top* and *bottom* nodes, respectively. Let $\mathcal{P}$ denote the set of maximal paths $p = [u_1^x, u_1, \ldots, u_k, u_k^y]$ in D_b. The paths in $\mathcal{P}$ are classified based on the position of their endpoints. A path p can be labeled as follows:

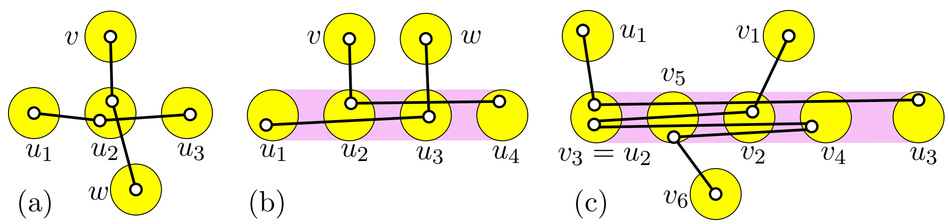


Figure 11: Three pairs of incompatible paths.

- *cross-chain* if u_1^x and u_k^y are top and bottom nodes respectively,
- *top chain* (resp., *bottom chain*) if both u_1^x and u_k^y are top nodes (resp., bottom nodes),
- *pin* if $p = [u_1^x, u_1, u_1^x]$ (note that every pin is a top or a bottom chain),
- *V-chain* if $p = [u_1^x, u_1, u_1^y]$, where $x \neq y$ and p is a top or a bottom chain.

Finally, let $\mathcal{P}in \subset \mathcal{P}$ be the set of pins, and $\mathcal{V} \subset \mathcal{P}$ the set of V-chains.

3.4 Clusters

As a preprocessing step for spur elimination (Section 5), we group all nodes that do not lie inside a bar into *clusters*. After node-expansion and new-bar-expansion, all such nodes lie on a boundary of a disk (circular or elliptical). For every sober node u, we create $\deg(u)$ clusters as follows. Refer to Figure 12. The node expansion has replaced u with new nodes on ∂D_u. Subdivide each segment in D_u with two new nodes. For each node $v \in \partial D_u$, form a cluster $C(v)$ that consists of v and all adjacent (subdivision) nodes inside D_u. For each node u on the boundary of an elliptical disk D_b, subdivide the unique edge outside D_b incident to u with a node u^*. Form a cluster $C(u^*)$ containing u and u^*. Every cluster maintains the following invariants.

Cluster Invariants. For every cluster $C(u)$:

(I1) $C(u)$ induces a tree $T[u]$ in the image graph rooted at u.

(I2) Every maximal path of P in $C(u)$ is of one of the following two types:

  (a) both endpoints are at the root of $T[u]$ and the path contains a single spur;
  (b) one endpoint is at the root, the other is at a leaf, and the path contains no spurs.

(I3) Every leaf node ℓ satisfies one of the following conditions:

  (a) ℓ has degree one in the image graph of P (and every vertex at ℓ is a spur);
  (b) ℓ has degree two in the image graph of P and there is no spur at ℓ.

(I4) No edge passes through a leaf ℓ (i.e., there is no edge $[a, b]$ such that $\ell \in ab$ but $\ell \notin \{a, b\}$).

Initially, every cluster trivially satisfies (I1)–(I2) and every leaf node satisfies (I3)–(I4) since it was created by a subdivision.

Dummy vertices. Although the operations described in Sections 4 and 5 introduce new nodes in the clusters, the image graph will always have $O(n)$ nodes and segments. A vertex at a cluster node is called a *benchmark* if it is a spur or if it is at a leaf node; otherwise it is called a *dummy vertex*. Paths traversing clusters may jointly contain $\Theta(n^2)$ dummy vertices in the worst case, however we

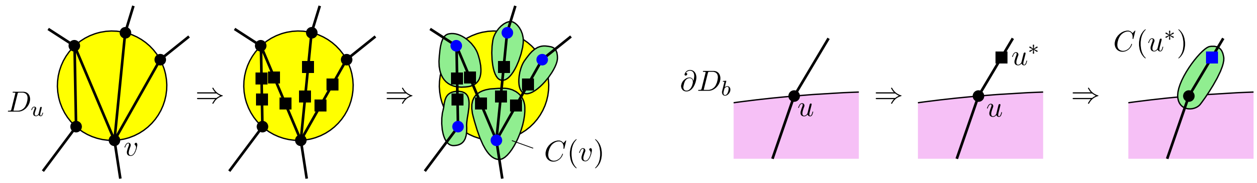


Figure 12: Formation of new clusters around (left) a sober node and (right) a node on the boundary of an elliptical disk. The roots of the induced trees are colored blue.

do not store these explicitly. By (I1), (I2), and (I3) a maximal path in a cluster can be uniquely encoded by one benchmark vertex: if it goes from a root to a spur at an interior node s and back, we record only $[s]$; and if it traverses $T[u]$ from the root to a leaf ℓ, we record only $[\ell]$.

4 Bar simplification

In this section we introduce three new ws-equivalent operations and show that they can eliminate all vertices from each bar independently (thus eliminating all forks). The bar decomposition is pre-computed, and the bars remain fixed during this phase (even though all edges along each bar are eliminated).

We give an overview of the overall effect of the operations (Section 4.1), define them and show that they are ws-equivalent (Sections 4.2–4.3), and then show how to use these operations to eliminate all vertices from a bar (Section 4.4).

4.1 Overview

After preprocessing in Section 3, we may assume that P has no edge crossings and satisfies (A1)–(A2). We summarize the overall effect of the bar simplification subroutine for a given expanded bar.

Changes in the image graph *G*. Refer to Figure 13. All nodes in the interior of the ellipse D_b are eliminated. Some spurs on b are moved to new nodes in the clusters along ∂D_b. Segments inside D_b connect two leaves of trees induced by clusters.

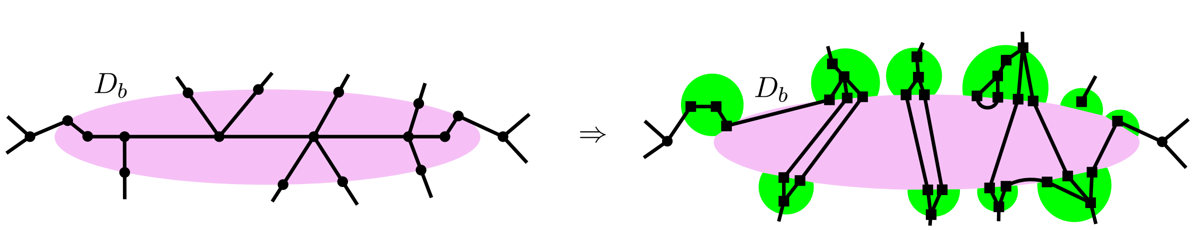


Figure 13: The changes in the image graph caused by a bar simplification.

Changes in the polygon *P*. Refer to Figure 14. Consider a maximal path p in P that lies in D_b. The bar simplification replaces $p = [u, \ldots, v]$ with a new path p'. By (I3)-(I4), only nodes u and v in p lie on ∂D_b. If p is the concatenation of a path p_1 and p_1^{-1} (the path formed by the vertices of p_1 in reverse order), then p' is a spur in the cluster containing u (Figure 14 (a)). If p has no such decomposition, but its two endpoints are at the same node, $u = v$, then p' is a single edge connecting two leaves in the cluster containing u (Figure 14 (b)). If the endpoints of p are at two different nodes, p' is an edge between two leaves of the clusters containing u and v respectively (Figure 14 (c) and (d)).

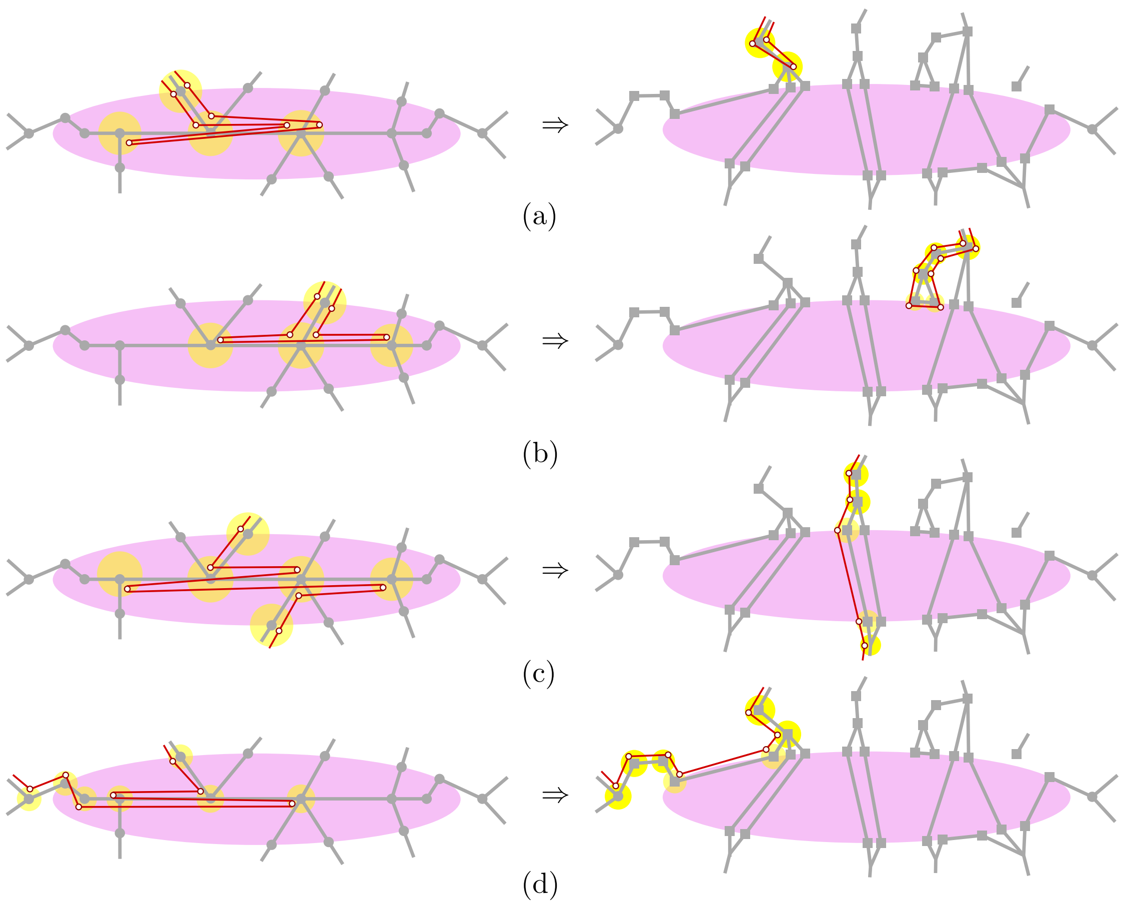


Figure 14: The changes in the polygon caused by a bar simplification.

4.2 Primitives

The operations in Section 4.3 rely on two basic steps, spur-reduction and node-split (see Figure 15). Together with merge and subdivision, these operations are called *primitives*.

spur-reduction(u, v)**.** Assume that every vertex at node u has at least one incident edge $[u, v]$. While there exists a path $[u, v, u]$, replace it with a single-vertex path $[u]$. (See Figure 15, left.)

node-split(u, v, w)**.** Assume that segments uv and vw are consecutive in radial order around v, node v is not in the interior of any edge that contains uv or vw; and P has no spurs of the form $[u, v, u]$ or $[w, v, w]$. Create node v^* in the interior of the wedge $\angle uvw$ sufficiently close to v; replace every path $[u, v, w]$ with $[u, v^*, w]$. (See Figure 15, right.)

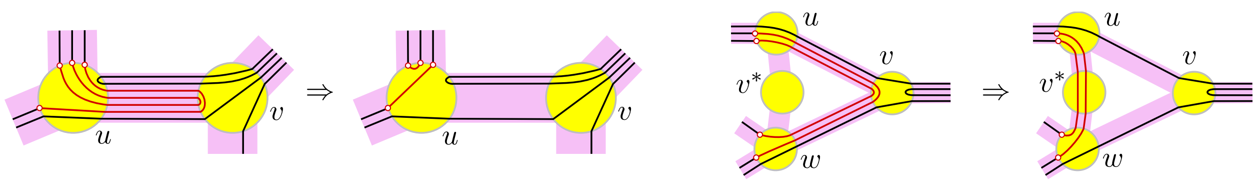


Figure 15: Left: Spur-reduction(u, v). Right: Node-split(u, v, w).

The following two lemmas are generalizations of the results in [6, Section 5].

Lemma 4. *Operation* spur-reduction *is ws-equivalent.*

Proof. Let P' be obtained from applying spur-reduction(u, v) to P. First suppose that P is weakly simple. Then, there exists a simple polygon $Q \in \Phi(P)$ represented by its signature. Successively replace any path $[u, v, u]$ by $[u]$ and delete these two edges from the ordering. The new signature defines a polygon Q' in the strip system of P'. By the assumption in the operation, every edge of Q in D_u is adjacent to an edge in N_{uv}, which has another endpoint in ∂D_v. Since Q is simple, the counterclockwise order of the endpoints of the deleted edges in ∂D_v is the same as the clockwise order of the endpoints of the new edges in ∂D_u. Thus, the new matching in D_u produces no crossings, $Q' \in \Phi(P')$, and P' is weakly simple.

Now suppose P' is weakly simple. Then, there exists a simple polygon $Q' \in \Phi(P')$ represented by its signature. Let H'_u be the set of all vertices in the node u in P'. Each vertex in H'_u corresponds to an edge in Q' that lies in the disk D_u; these edges are noncrossing chords of the circle ∂D_u. We define a partial ordering on H'_u: For two vertices $u_1, u_2 \in H'_u$, let $u_1 \prec u_2$ if the chord corresponding to u_1 separates the chord of u_2 from N_{uv} within the disk D_u. Intuitively, we have $u_1 \prec u_2$ if u_1 blocks u_2 from the corridor N_{uv}. Note that if $u_1 \prec u_2$, then neither endpoint of the chord corresponding to u_1 is on the boundary of N_{uv}; consequently u_1 was obtained from a path $[u, v, u]$ or $[u, v, u, v, u, \dots, u]$ in P after removing one or more spurs. We expand the paths $u_i \in H'_u$ incrementally, in an order determined by any linear extension of the partial ordering $\prec$. Replace the first vertex $u_1 \in H'_u$ by $[p, u, v, u]$ (or $[p, u, v, u, v, u, \dots, u, q]$ if needed), and modify the signature by inserting consecutive new edges into the total order of the edges along uv at any position that is not separated from the chord in D_u that corresponds to u_1. The resulting polygon P'' and the new signature define a polygon Q'' in the strip system of P''. By construction, the new edges in D_v connect consecutive endpoints in counterclockwise order around v, thus the new matching in D_v is noncrossing. In the disk D_u, the operation replaces the chord corresponding to u_1 by noncrossing new chords. Each new edge in D_u has at least one endpoint in N_{uv}; consequently, none of them blocks access to $N_{u,v}$. Then, the new matching in D_u has no crossing and $Q'' \in \Phi(P'')$. By repeating this procedure we obtain P and a simple polygon $Q \in \Phi(P)$, hence P is weakly simple. ☐

Lemma 5. *Operation* node-split *is ws-equivalent.*

Proof. Let P' be obtained from P via node-split(u, v, w). First assume that P is weakly simple. Then there is a simple polygon $Q \in \Phi(P)$. Consider the clockwise order of edges around v. Since Q is simple, the order of the edges $[u, v]$ of paths $[u, v, w]$ must be the reverse order of its adjacent edges $[v, w]$ (the paths must be nested as shown in Figure 15(right)). Because P has no spurs of the form $[u, v, u]$ or $[w, v, w]$, and the edges of P that pass through v avoid both uv and vw, every edge between a pair of adjacent edges $[u, v]$ and $[v, w]$ is also part of a path $[u, v, w]$. Replace the paths $[u, v, w]$ by $[u, v^*, w]$ and set the order of edges at segments uv^* and v^*w to be the same order of the removed edges at uv and vw. This defines a polygon $Q' \in \Phi(P')$, which is simple because the circular order of endpoints around D_u and D_w remains unchanged and the matching in D_{v^*} is a subset of the matching in D_v.

Now, assume that P' is weakly simple. Since the face in the image graph bounded by u, v, w, v^* is empty, we can change the embedding of the graph by bringing v^* arbitrarily close to v, maintaining weak simplicity. Let δ be the distance between v^* and v. Let $Q' \in \Phi(P')$ be a simple polygon defined on disks of radius ε. Then, Q' is within $\varepsilon + \delta$ Fréchet distance from P and therefore P is weakly simple. ☐

4.3 Operations

We describe three complex operations: pin-extraction, V-shortcut, and L-shortcut. In Section 4.4, we show how to use them to eliminate spurs along any given bar b. The pin-extraction and V-shortcut operations eliminate pins and V-chains. Chains in $\mathcal{P}$ with two or more vertices in the interior of D_b are simplified incrementally, removing one vertex at a time, by the L-shortcut operation.

Since the image graph is determined by the polygon, it would suffice to describe how the operations modify the polygon. However, it is sometimes more convenient to first define new nodes and segments in the image graph, and use them to describe the changes in the polygon. In the last step of these operations, we remove any node (segment) that contains no vertex (edge), to ensure that the image graph is consistent with the polygon.

> pin-extraction(u, v)**.** Assume that P satisfies (I1)–(I4) and contains a pin $[v, u, v] \in \mathcal{P}in$. By (I3), node v is adjacent to a unique node w outside of D_b. Perform the following three primitives: (1) subdivision of every path $[v, w]$ into $[v, w^*, w]$; (2) spur-reduction(v, u). (3) spur-reduction(w^*, v). (4) Update the image graph. See Figure 16 for an example.

> V-shortcut(v_1, u, v_2)**.** Assume that P satisfies (I1)–(I4) and $[v_1, u, v_2] \in \mathcal{V}$. Furthermore, P contains no pin of the form $[v_1, u, v_1]$ or $[v_2, u, v_2]$, and no edge $[u, q]$ such that segment uq is in the interior of the wedge $\angle v_1 u v_2$. By (I3), nodes v_1 and v_2 are each adjacent to unique nodes w_1 and w_2 outside of D_b, respectively.
>
> The operation executes the following primitives sequentially: (1) node-split(v_1, u, v_2), which creates a temporary node u^*; (2) node-split(u^*, v_1, w_1) and node-split(u^*, v_2, w_2); which create $v_1^*, v_2^* \in \partial D_b$, respectively; (3) merge every path $[v_1^*, u^*, v_2^*]$ to $[v_1^*, v_2^*]$. (4) Update the image graph. See Figure 17 for an example.

Lemma 6. pin-extraction *and* V-shortcut *are ws-equivalent and maintain (A1)–(A2) in D_b and (I1)–(I4) in adjacent clusters.*

Proof. pin-extraction**.** By construction, the operation maintains (A1)–(A2) in D_b and (I1)–(I4) in adjacent clusters. Also, (I3)–(I4) ensure that spur-reduction(v, u) in step (2) satisfies its preconditions. Consequently, all three primitives are ws-equivalent.

V-shortcut**.** By construction, the operation maintains (A1)–(A2) in D_b and (I1)–(I4) in adjacent clusters. The first two primitives are ws-equivalent by Lemma 5. The third step is ws-equivalent because triangle $\Delta(u^* v_1^* v_2^*)$ is empty of nodes and segments, by assumption. ☐

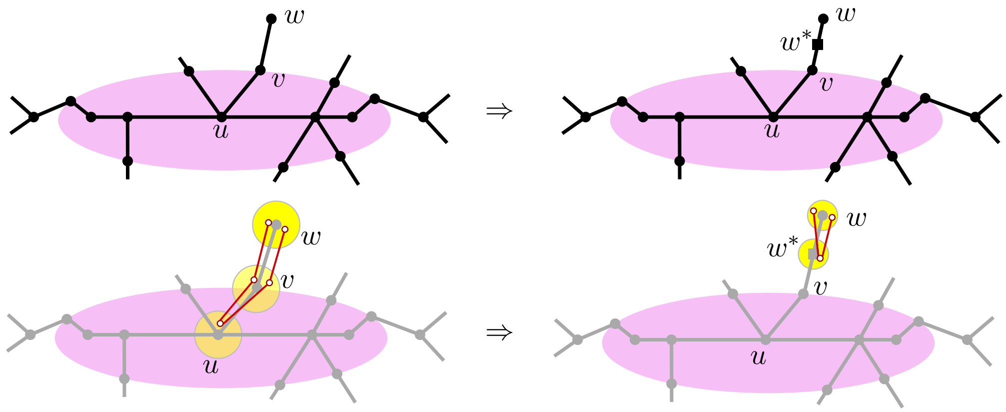


Figure 16: pin-extraction. Changes in the image graph (top), changes in the polygon (bottom).

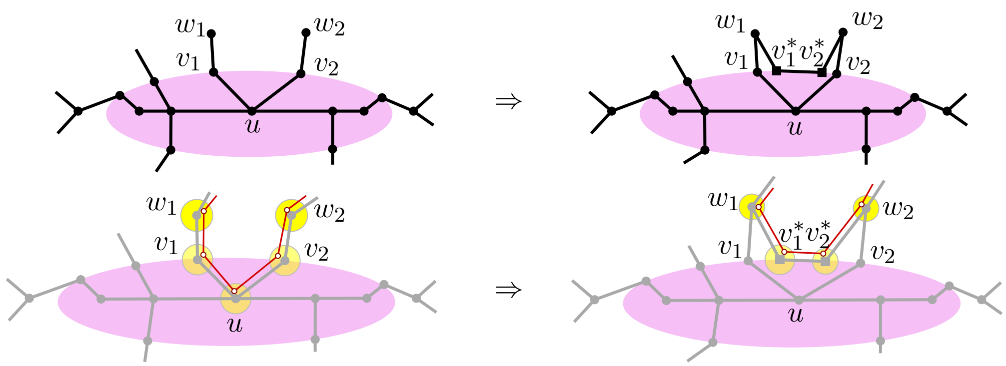


Figure 17: V-shortcut. Changes in the image graph (top), changes in the polygon (bottom).

L-shortcut **operation.** The purpose of this operation is to eliminate a vertex of a path that has an edge along a given bar. Before describing the operation, we introduce some notation; refer to Figure 18. For a node $v \in \partial D_b$, let L_v be the set of paths $[v, u_1, u_2]$ in P such that $u_1, u_2 \in \text{int}(D_b)$. Each path in $\mathcal{P}$ is either in $\mathcal{P}in$, in $\mathcal{V}$, or has two subpaths in some L_v. Let M_{cr} be the set of longest edges of cross-chains in $\mathcal{P}$. Denote by $\widehat{L_v} \subset L_v$ the set of paths $[v, u_1, u_2]$, where $[u_1, u_2]$ is *not* in M_{cr}.

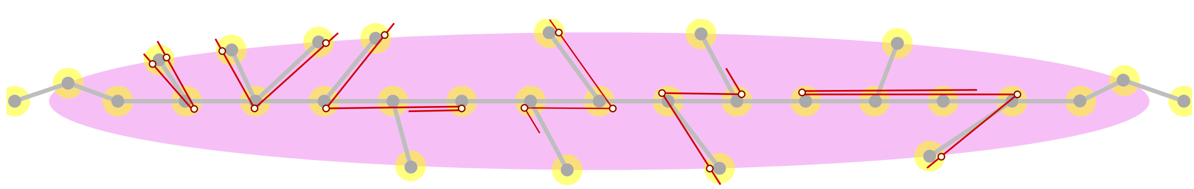


Figure 18: Paths in $\mathcal{P}in$, $\mathcal{V}$, L_v^{TR}, L_v^{TL}, L_v^{BR}, and L_v^{BL}.

We partition L_v into four subsets (refer to Figure 18): a path $[v, u_1, u_2] \in L_v$ is in

1. L_v^{TR} (*top-right*) if v is a *top* vertex and $x(u_1) < x(u_2)$;
2. L_v^{TL} (*top-left*) if v is a *top* vertex and $x(u_1) > x(u_2)$;
3. L_v^{BR} (*bottom-right*) if v is a *bottom* vertex and $x(u_1) < x(u_2)$;

4. L_v^{BL} (*bottom-left*) if v is a *bottom* vertex and $x(u_1) < x(u_2)$.

We partition $\widehat{L_v}$ into four subsets analogously. We define the operation L-shortcut for paths in L_v^{TR}; the definition for the other subsets can be obtained by suitable reflections.

L-shortcut(v, TR)**.** Assume that P satisfies (I1)–(I4), $v \in \partial D_b$ and $L_v^{TR} \neq \emptyset$. By (I3), v is adjacent to a unique node $u_1 \in b$ and to a unique node $w \notin D_b$. Let U denote the set of all nodes u_2 for which $[v, u_1, u_2] \in L_v^{TR}$. Let $u_{\min} \in U$ and $u_{\max} \in U$ be the leftmost and rightmost node in U, respectively. Further assume that P satisfies:

(B1) there is no pin of the form $[v, u_1, v]$;
(B2) no edge $[p, u_1]$ such that segment pu_1 is in the interior of the wedge $\angle v u_1 u_{\min}$;
(B3) no edge $[p, q]$ such that $p \in \partial D_b$ is a top vertex and $q \in b$, $x(u_1) < x(q) < x(u_{\max})$.

Do the following (see Figure 19 for an example).

(0) Create a new node $v^* \in \partial D_b$ to the right of v sufficiently close to v.
(1) For every path $[v, u_1, u_2] \in L_v^{TR}$ in which u_1u_2 is the *only* longest edge of a cross-chain, create a crimp by replacing $[u_1, u_2]$ with $[u_1, u_2, u_1, u_2]$.
(2) Replace every path $[w, v, u_1, u_{\min}]$ by $[w, v^*, u_{\min}]$.
(3) Replace every path $[w, v, u_1, u_2]$, where $u_2 \in U$ and $u_2 \neq u_{\min}$, by $[w, v^*, u_{\min}, u_2]$.
(4) Update the image graph.

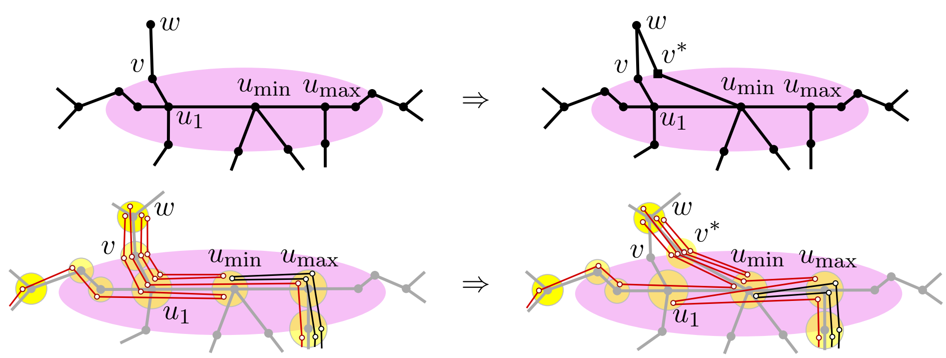


Figure 19: L-shortcut. Changes in the image graph (top), changes in the polygon (bottom).

See Figure 20 for an explanation of why L-shortcut requires conditions (B2)–(B3) and phase (1) of the operation. If we omit any of these conditions, L-shortcut would not be ws-equivalent.

Lemma 7. L-shortcut *is ws-equivalent and maintains (A1)–(A2) in D_b and (I1)–(I4) in adjacent clusters.*

Proof. Let P_1 be the polygon obtained from P after phase (1) of L-shortcut(v, TR) and P_2 be the polygon obtained after phase (3). Note that phase (1) of the operation only creates crimps, and it is ws-equivalent by Lemma 1. Let H be the set of edges $[u_1, u_2]$ of paths $[v, u_1, u_2] \in L_v^{TR}$. Phases (2)–(3) are equivalent to the concatenation of the primitives: subdivision, node-split, and merge. Specifically, they are equivalent to subdividing every edge in H into $[u_1, u_{\min}, u_2]$ whenever $u_2 \neq u_{\min}$, and applying node-split$(v, u_1, u_{\min})$ (which creates u_1^*) to P_2 followed by node-split(w, v, u_1^*)

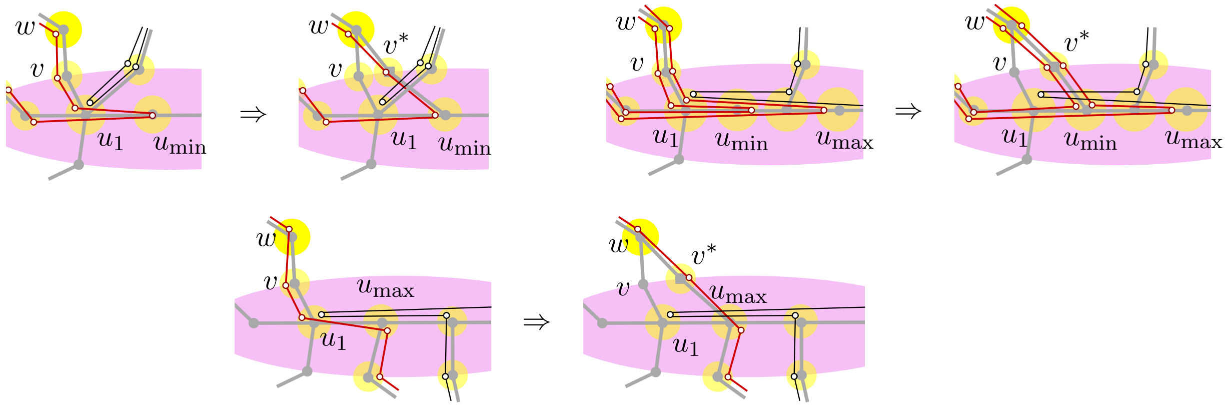


Figure 20: Cases in which L-shortcut is not ws-equivalent. Top left: P does not satisfy (B2). Top right: P does not satisfy (B3). Bottom: the operation skips phase (1).

(which creates v^*), and merging every path $[v^*, u_1^*, u_{\min}]$ to $[v^*, u_{\min}]$. The only primitive that may not satisfy its preconditions is node-split$(v, u_1, u_{\min})$: segment $u_1u_{\min}$ may be collinear with several segments of b, and P_2 may contain spurs that overlap with $u_1u_{\min}$. In the next paragraph, we show that the spurs that may overlap with $u_1u_{\min}$ do not pose a problem, and we can essentially repeat the proof of Lemma 5.

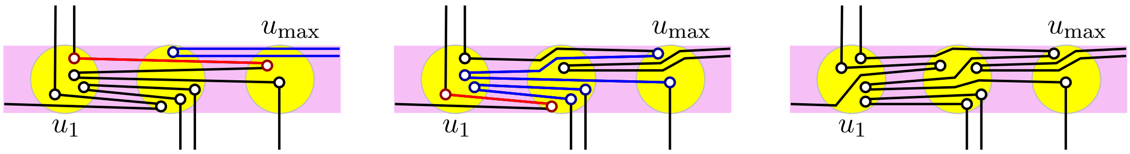
Assume that P_1 is weakly simple and consider a polygon $Q_1 \in \Phi(P_1)$. Due to (A1)–(A2) and phase (1), every path in L_v^{TR} is a sub-path of some path $[v, u_1, u_2, u_3]$ where $x(u_3) \leq x(u_2)$. We show that P_1 has a perturbation in $\Phi(P_1)$ with the following property:

($\star$) Every edge $[u_1, u_2] \in H$ lies above all overlapping edges $e \notin H$.

Let $Q_1 \in \Phi(P_1)$ be a perturbation of P_1 into a simple polygon that has the minimum number of edges $[u_1, u_2] \in H$ that violate ($\star$). We claim that Q_1 satisfies ($\star$). Suppose the contrary, that Q_1 does not satisfy ($\star$). For a contradiction, we modify $Q_1 \in \Phi(P_1)$ and obtain another perturbation $Q_1' \in \Phi(P_1)$ that has strictly fewer edges that violate ($\star$) as shown in Figure 21. Recall that Q_1 yields a total order of edges in each segment of b based on the above-below relationship. Let $[u_1, u_2'] \in H$ be the highest edge that violates ($\star$), and assume that this edge is part of a path $[v, u_1, u_2', u_3']$. Let Z be the set of edges that are above $[u_1, u_2']$ within the corridors between u_1 and u_2', and are not in H. By (B2)–(B3) and Lemma 2, every edge $[z_1, z_2] \in Z$ must be part of a path $[z_1, z_2, z_3]$ where $x(u_1) \leq x(z_2) < x(u_2') \leq x(z_1)$ and $x(u_2') \leq x(z_3)$, otherwise Q_1 would not be simple. We modify $\sigma(Q_1)$ by moving the edges in Z, maintaining their relative order, immediately below edge $[u_2', u_3']$ in all segments between u_1 and u_2'. This results in a simple polygon $Q_1' \in \Phi(P_1)$ such that $[u_1, u_2']$ and all edges in H above $[u_1, u_2']$ satisfy ($\star$), contradicting the choice of Q_1.

We can proceed as in the proof of Lemma 5, using a perturbation $Q_1 \in \Phi(P_1)$ that satisfies ($\star$) to show that P_2 is weakly simple if and only if P_1 is weakly simple, that is, phases (2)–(3) are ws-equivalent.

By construction, (I1)–(I4) are maintained. Note that the intermediate polygon P_1 may violate condition (A2), since phase (1) introduces crimps. However, after phase (3), conditions (A1) and (A2) are restored, and operation L-shortcut maintains (A1)–(A2) in the ellipse D_b. □



(a) (b) (c)

Figure 21: (a) A perturbation Q_1 that violates property ($\star$); the highest edge $[u_1, u_2'] \in H$ that violates ($\star$) is red, and edges in Z are blue. (b) We can modify Q_1 to reduce the number of edges in H that violate ($\star$). (c) There exists a perturbation Q_1 that satisfies ($\star$).

4.4 Bar simplification algorithm

In this section, we describe an algorithm, called bar-simplification, that incrementally removes all spurs of the polygon P from a bar b using a sequence of pin-extraction, V-shortcut, and L-shortcut operations. Informally, our algorithm "unwinds" each polygonal chain in the bar. It extracts pins and V-chains whenever possible. Any other chain in D_b contains edges along bar b, and the sequence of these edge lengths is unimodal (cf. Lemma 2). Our algorithm "unwinds" these chains by a sequence of L-shortcut operations. Each operation eliminates or reduces one of the shortest edges along b (see Figure 22). The algorithm alternates between L-shortcut(v, TR) and L-shortcut(v, TL) to unwind the chains from their top endpoints to the longest edge in b; and then uses L-shortcut(v, BR) and L-shortcut(v, BL) to resolve the bottom part.

When we unwind the chains in D_b starting from their top vertices using L-shortcut(v, TR) and L-shortcut(v, TL), we cannot hope to remove the longest edge of a cross-chain. We stop using the operations when every path in L_v^{TR} contains a longest edge of a cross-chain. This motivates the use of $\widehat{L}_v^{TR}$ (instead of L_v^{TR}) in step (iii) below. We continue with the algorithm and its analysis.

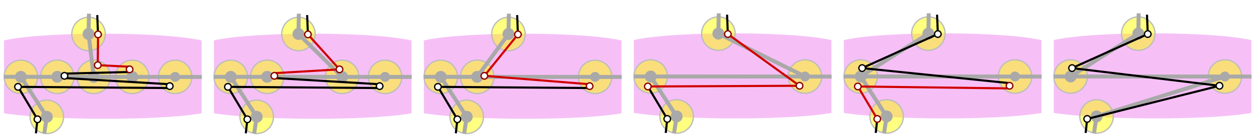


Figure 22: Life cycle of a cross-chain in the while loop of bar-simplification. The steps applied, from left to right, are: (iii), (iv), (iii), (iv), and (vi).

Algorithm bar-simplification(P, b).
While P has an edge along b, perform one operation as follows.

(i) If $\mathcal{P}in \neq \emptyset$, pick an arbitrary pin $[v, u, v]$ and perform pin-extraction(u, v).

(ii) Else if $\mathcal{V} \neq \emptyset$, then let $[v_1, u, v_2] \in \mathcal{V}$ be a path where $|x(v_1) - x(v_2)|$ is minimal. If there is no segment uq in the wedge $\angle v_1 u v_2$, perform V-shortcut(v_1, u, v_2), else report that P is not weakly simple and halt.

(iii) Else if there exists $v \in \partial D_b$ such that $\widehat{L}_v^{TR} \neq \emptyset$, do:

  (a) Let v be the rightmost node where $L_v^{TR} \neq \emptyset$.

  (b) If L_v^{TR} satisfies (B2)–(B3), do L-shortcut(v, TR).

  (c) Else let v' be the leftmost node such that $x(v) < x(v')$ and $L_{v'}^{TL} \neq \emptyset$, or record that no such vertex v' exists.

(c.1) If v' does not exist, or $L_{v'}^{TL}$ does not satisfy (B2)–(B3), or any path in $L_{v'}^{TL}$ contains a longest edge of a cross-chain, then report that P is not weakly simple and halt.

(c.2) Else do L-shortcut(v', TL).

(iv) Else if there exists $v \in \partial D_b$ such that $L_v^{TL} \neq \emptyset$, perform steps (iii)a–(iii)c with left–right and TR–TL interchanged. (Note the use of L_v instead of $\widehat{L_v}$. The same applies to (vi) below).

(v) Else if there exists $v \in \partial D_b$ such that $\widehat{L}_v^{BL} \neq \emptyset$, perform steps (iii)a–(iii)c using BL and BR in place of TR and TL, respectively, and left-right interchanged.

(vi) Else if there exists $v \in \partial D_b$ such that $L_v^{BR} \neq \emptyset$, perform steps (iii)a–(iii)c using BR and BL in place of TR and TL, respectively.

(vii) Else invoke old-bar-expansion.

Return P (end of algorithm).

Lemma 8. *The operations performed by* bar-simplification(P, b) *are ws-equivalent, and maintain properties (A1)–(A2) in D_b and (I1)–(I4) in adjacent clusters. The algorithm either removes all nodes from the ellipse D_b, or reports that P is not weakly simple. The* L-shortcut *operations performed by the algorithm create at most two crimps in each cross-chain in $\mathcal{P}$.*

Proof. We show that the algorithm only uses operations that satisfy their preconditions, and reports that P is not weakly simple only when P contains a forbidden configuration.

Steps (i)–(ii). Since every pin can be extracted from a polygon satisfying (I1)–(I4), we may assume that $\mathcal{P}in = \emptyset$. Suppose that $\mathcal{V} \neq \emptyset$. Let $[v_1, u, v_2] \in \mathcal{V}$ be a V-chain such that $|x(v_1) - x(v_2)|$ is minimal. Since $\mathcal{P}in = \emptyset$, the only obstacle for the precondition of V-shortcut is an edge $[u, q]$ such that segment uq is in the interior of the wedge $\angle v_1 u v_2$ (or else the image graph would have a crossing). If such an edge exists, it is part of a path $[p, u, q]$. The node q is in ∂D_b between v_1 and v_2. Note that $p \neq q$, otherwise $[p, u, q]$ would be a pin. Further, p cannot be a node in the interior of the wedge $\angle v_1 u v_2$, otherwise $[p, u, q]$ would be a V-chain where $|x(p) - x(q)| < |x(v_1) - x(v_2)|$, contrary to the choice of $[v_1, u, v_2] \in \mathcal{V}$. Consequently, p must be in the exterior of the wedge $\angle v_1 u v_2$. In this case, the paths $[v_1, u, v_2]$ and $[p, u, q]$ form the forbidden configuration in Corollary 1(1), and the algorithm correctly reports that P is not weakly simple. If no such edge $[u, q]$ exists, then V-shortcut(v_1, u, v_2) satisfies all preconditions and it is ws-equivalent by Lemma 6. Henceforth, we may assume that $\mathcal{P}in = \emptyset$ and $\mathcal{V} = \emptyset$.

Step (iii)–(iv). By symmetry, we consider only step (iii). Since $\mathcal{P}in = \emptyset$, condition (B1) is met. In step (iii)b, if (B2)-(B3) are also satisfied, then L-shortcut(v, TR) is ws-equivalent by Lemma 7. If condition (B2) or (B3) fails, we proceed with step (iii)c.

Step (iii)(c.1). We show that in these cases the algorithm correctly reports that P is not weakly simple. Assume first that v' does not exist. Since L_v^{TR} does not satisfy (B2) or (B3), there exists an edge $[p, q]$ such that $x(u_1) \leq x(q) < x(u_{\max})$ and $p \in \partial D_b$ is a top node. Edge $[p, q]$ is part of some path $[p, q, r]$. Note that r cannot be a top vertex of ∂D_b, since $\mathcal{P}in = \emptyset$ and $\mathcal{V} = \emptyset$. If r is on b and $x(q) < x(r)$, then $[p, q, r] \in L_p^{TR}$, which contradicts the choice of node v. If r is on b and $x(r) < x(q)$, then $[p, q, r] \in L_p^{TL}$ and v' exists. It follows that r is a bottom vertex, and then the paths $[v, u_1, u_{\max}]$ and $[p, q, r]$ form a forbidden configuration in Corollary 1(1) or (3).

Assume now that v' exists but $L_{v'}^{TL}$ does not satisfy (B2) or (B3). Let $[v', u_1', u_{\max}']$ be the path in $L_{v'}^{TL}$ with the longest edge on b. By the definitions of (B2)–(B3), $x(u_1) \leq x(u_1') < x(u_{\max})$.

If $x(u'_{\max}) < x(u_1)$, then $[v, u_1, u_{\max}]$ and $[v', u'_1, u'_{\max}]$ form the forbidden configuration in Corollary 1(2). Else, we have $x(u_1) \le x(u'_{\max}) < x(u'_1) < x(u_{\max})$. This implies that any edge $[p, q]$ that violates (B2) or (B3) for $L^{TL}_{v'}$ must also violate (B2) or (B3) for L^{TR}_v. However, this contradicts the choice of v (rightmost where $L^{TR}_v \neq \emptyset$) and v' (leftmost, $x(v) < x(v')$, where $L^{TL}_{v'} \neq \emptyset$).

Next assume that there is a path $[v', u'_1, u'_2] \in L^{TL}_{v'}$ such that $[u'_1, u'_2]$ is the longest edge of a cross-chain. Then this cross-chain is of the form $[v', u'_1, u'_2, \ldots, p']$, where all interior vertices lie on the line segment $u'_1 u'_2$, and p' is a bottom vertex. Now $[v, u_1, u_{\max}]$ and this cross-chain form the forbidden configuration in Corollary 1(3). In all three cases in step (iii)(c.1), the algorithm correctly reports that P is not weakly simple.

Step (iii)(c.2). Let the path $[v', u'_1, u'_{\max}] \in L^{TL}_{v'}$ be selected in L-shortcut(v', TL) by the algorithm. Since conditions (B1)–(B3) are satisfied, L-shortcut(v', TL) is ws-equivalent by Lemma 7.

Steps (v)–(vii). If steps (i)–(iv) do not apply, then $\widehat{L}^{TR}_v \cup L^{TL}_v = \emptyset$. That is, for every path $[v, u_1, u_2] \in L^{TR}$, we have $[u_1, u_2] \in M_{cr}$. In particular, there are no top chains. The operations in (v)–(vi) do not change these properties. Consequently, once steps (v)–(vi) are executed for the first time, steps (iii)–(iv) are never executed again. By a symmetric argument, steps (v)–(vi) eliminate all paths in $\widehat{L}^{BL}_v \cup L^{BR}_v$. When the algorithm reaches step (vii), every edge in b is necessarily in M_{cr} and $L^{TL}_v \cup L^{BR}_v = \emptyset$. Consequently, by Lemma 2, b contains no spurs and old-bar-expansion is ws-equivalent. This operation eliminates all nodes in the interior of D_b.

Termination. Each pin-extraction and V-shortcut operation reduces the number of vertices of P within D_b. Operation L-shortcut(v, X), $X \in \{TR, TL, BR, BL\}$, either reduces the number of interior vertices, or produces a crimp if edge $[u_1, u_2]$ is a longest edge of a cross-chain. For termination, it is enough to show that, for each cross-chain $c \in \mathcal{P}$, the algorithm introduces a crimp at most once in steps (iii)–(iv), and at most once in steps (v)–(vi). Without loss of generality, consider step (iii).

Note that step (iii) may apply an L-shortcut operation in two possible cases: (iii)b and (iii)c. However, an L-shortcut operation in (iii)c does not create crimps: L-shortcut is performed when all three conditions in (iii)(c.1) fail. In this case, $L^{TR}_{v'}$ does not contain any edge in M_{cr}, and L-shortcut does not create crimps. We may assume that step (iii) creates crimps in case (iii)b only.

Every cross-chain remains a cross-chain in algorithm bar-simplification: operations pin-extraction and V-shortcut do not modify cross-chains; and operations L-shortcut and old-bar-expansion modify only the first or last few edges of a cross-chain. A longest edge of a cross-chain c always connects the same two nodes in b until step (vii) (old-bar-expansion), although the *number* of longest edges in c may change. When L-shortcut(v, X) modifies a cross-chain, it moves its endpoint from $v \in \partial D_b$ to a nearby new node $v^* \in \partial D$. Consequently, if L^X_v, $X \in \{TR, TL\}$ contains the first two edges of two chains in $\mathcal{P}$, then they have been modified by the same sequence of previous L-shortcut operations.

Suppose, for contradiction, that two invocations of step (iii)b create crimps in a cross-chain c, say, in operations L-shortcut(v_0, TR) and L-shortcut(v_2, TR) (see Figure 23). The first invocation replaces $[v_0, u_1, u_2]$ with $[v^*_0, u_{\min}, u_2, u_1, u_2]$ (where the edge $[u_{\min}, u_2]$ may vanish if $u_{\min} = u_2$). The resulting cross-chain has two maximal longest edges, $[u_2, u_1]$ and $[u_1, u_2]$. Since L-shortcut creates crimps only if the longest edge is unique, there must be an intermediate operation L-shortcut(v_1, TL) that removes or shortens the edge $[u_2, u_1]$, so that $[u_1, u_2]$ becomes the unique longest edge again. When L-shortcut(v_1, TL) is performed in a step (iv), we have $\widehat{L}^{TR}_v = \emptyset$ for all top nodes v, and $L^{TL}_{v'} = \emptyset$ for all top nodes v', $x(v') < x(v_1)$. The steps between L-shortcut(v_1, TL) and L-shortcut(v_2, TR) modify only cross-chains whose top node is at or to the right of the top node of c (L-shortcut operations move the top vertex of c to the left, from v_1 to v_2 in one or more steps). Consequently, when L-shortcut(v_2, TR) is performed in a step (iii), we still have $\widehat{L}^{TR}_{v'} = L^{TL}_{v'} = \emptyset$ for all top nodes v', $x(v') < x(v_2)$.

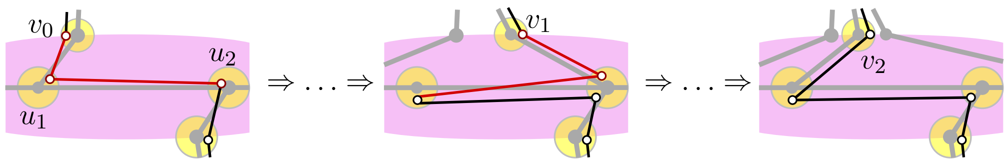


Figure 23: At most one crimp can be created in a cross-chain by steps (iii)b.

When L-shortcut(v_2, TR) is performed, we have $[v_2, u_1, u_2] \in L^{TR}_{v_2}$ but $[v_2, u_1, u_2] \notin \widehat{L}^{TR}_{v_2}$ (since u_1u_2 is the longest edge of c). Step (iii) is performed only if $\widehat{L}^{TR}_{p} \neq \emptyset$ for some top vertex p. Since the rightmost top vertex where $L^{TR}_{v} \neq \emptyset$ is $v = v_2$, we have $x(p) \leq x(v_2)$. This implies $p = v_2$. Consequently there exists a chain $c' \in \mathcal{P}$ that contains a subpath $[v_2, u_1, u_3] \in L^{TR}_{v_2}$, such that $[u_1, u_3]$ is not the longest edge of c'. Since $L^{TR}_{v_2}$ contains the first two edges of both c and c', they have been modified by the same sequence of L-shortcut operations. Therefore c' contained $[v, u_1, u_2, u_1]$ initially. By Lemma 2, only the longest edge can repeat, hence $[u_1, u_2]$ is the longest edge of c'. This implies that $u_3 = u_2$ and $\widehat{L}^{TR}_{v_2} = \emptyset$, contradicting the condition in Step (iii).

We conclude that bar-simplification(P, b) introduces a crimp at most once in steps (iii)–(iv), and at most once in steps (v)–(vi) in each cross-chain. Since all other steps decrease the number of vertices in D_b, the algorithm terminates, as claimed. □

Lemma 9. *Algorithm* bar-simplification(P, b) *takes* $O(m \log m)$ *time using suitable data structures, where* m *is the number of vertices in* b.

Proof. Operations pin-extraction, V-shortcut, and L-shortcut each make $O(1)$ changes in the image graph. Operations pin-extraction and V-shortcut decrease the number of vertices inside D_b. Each L-shortcut does as well, except for the steps that create crimps. By Lemma 7, L-shortcut operations may create at most $2|\mathcal{P}| = O(m)$ crimps. So the total number of operations is $O(m)$.

When $[v, u_1, u_2] \in L^{TR}_{v}$ and $u_2 \neq u_{\min}$, L-shortcut replaces $[v, u_1, u_2]$ by $[v^*, u_{\min}, u_2]$: vertex $[u_1]$ shifts to $[u_2]$, but no vertex is eliminated. In the worst case, one L-shortcut modifies $\Theta(m)$ paths, so in $\Theta(m)$ operations the total number of vertex shifts is $\Theta(m^2)$.

Data structures. We maintain a cyclic list of nodes in ∂D_b given by the combinatorial embedding of the image graph. Since each operation adds a constant number of nodes to ∂D_b at positions adjacent to the nodes to which the operation was applied, such a list can be maintained using $O(1)$ time per operation. Our implementation does not maintain the paths in $\mathcal{P}$ explicitly. Instead, we use set operations. We maintain the sets $\mathcal{P}in$, $\mathcal{V}$, and L^{X}_{v}, with $v \in \partial D_b$ and $X \in \{TR, TL, BR, BL\}$, in sorted lists. The pins $[v, u, v] \in \mathcal{P}in$ are sorted by $x(v)$; the wedges $[v_1, u, v_2] \in \mathcal{V}$ are sorted by $|x(v_1) - x(v_2)|$. In every set L^{X}_{v}, the first two nodes in the paths $[v, u_1, u_2] \in L^{X}_{v}$ are the same by (I3)b, and so it is enough to store vertex $[u_2]$; these vertices are stored in a list sorted by $x(u_2)$. We also maintain binary variables to indicate for each path $[v, u_1, u_2] \in L^{X}_{v}$ whether it is part of a cross-chain, and whether $[u_1, u_2]$ is the only longest edge of that chain.

Running time analysis. The condition in step (ii) can be tested in $O(1)$ time by checking whether uv_1 and uv_2 are consecutive segments in the rotation of node u in the image graph. Steps (i)-(ii) remove pins and V-chains, taking linear time in the number of removed vertices, without introducing any path in any set.

Consider L-shortcut(v, TR), executed in step (iii), which can be generalized to other occurrences of the L-shortcut operation performed in one of steps (iii)–(vi). Recall that $\mathcal{P}in = \mathcal{V} = \emptyset$. Let p' be the leftmost top vertex in ∂D_b to the right of v, which can be found in $O(1)$ time using the cyclic list of nodes in ∂D_b. By (I3)b, every path $[p', q', r'] \in L^X_{p'}, X \in \{TR, TL\}$, must contain the edge $[p', q']$. If $x(q') < x(u_{\max})$, then (B2) or (B3) are not satisfied. Assume that $x(q') \geq x(u_{\max})$ and (B2) (resp., (B3)) is not satisfied. Then there must exist an edge $[p, u_1]$ (resp., $[p, q]$ where $x(q) < x(u_{\max})$) such that p is to the right of p'. Then, segments $p'q'$ and pu_1 (resp., pq) properly cross. This is a contradiction since no operation introduces crossings in the image graph. Hence (B2)–(B3) are satisfied if and only if either p' does not exist (i.e., v is the rightmost top vertex), or $x(u_{\max}) \leq x(q')$; this can be tested in $O(1)$ time. The elements $[v, u_1, u_{\min}] \in L^{TR}_v$ are simplified to $[v^*, u_{\min}]$. Consider one of these paths, and assume that the next edge along P is $[u_{\min}, u_3]$. Then, the path $[v^*, u_{\min}, u_3]$ is inserted into either $\mathcal{P}in \cup \mathcal{V}$ if $u_3 \in \partial D_b$ is a top vertex, or $L^{TL}_{v^*}$ if $u_3 \in b$. We can find each chain $[v, u_1, u_{\min}] \in L^{TR}_v$ in $O(1)$ time since L^{TR}_v is sorted by $x(u_2)$. Finally, all other paths of the form $[v, u_1, u_2] \in L^{TR}_v$, where $u_2 \neq u_{\min}$, become $[v^*, u_{\min}, u_2]$ and they form the new set $L^{TR}_{v^*}$. Since we store only the last vertex $[u_2]$, which is unchanged, we create $L^{TR}_{v^*}$ at no cost.

This representation allows the manipulation of $O(m)$ vertices with one set operation. The number of insert and delete operations in the sorted lists is proportional to the number of vertices that are removed from the interior of D_b, which is $O(m)$. Each insertion and deletion takes $O(\log m)$ time, and the overall time complexity is $O(m \log m)$. ☐

5 Spur elimination algorithm

After bar-simplification (Section 4), we obtain a polygon that has no forks and every spur is at an interior node of some cluster (formed on the boundary of some ellipse D_b). In the absence of forks, we can decide weak simplicity using [6, Theorem 5.1], but a naïve implementation runs in $O(n^2 \log n)$ time: successive applications of spur-reduction would perform an operation at each dummy vertex. In this section, we show how to eliminate spurs in $O(n \log n)$ time.

Formation of Groups. We create *groups* by gluing pairs of clusters with adjacent roots together. Recall that by (I1) each cluster induces a tree. We modify the image graph, transforming each tree in a cluster into a binary tree using ws-equivalent primitives. For each node s with more than two children, let s_1 and s_2 be the first two children in counterclockwise order. Create new nodes s'_1 and s'_2 by subdivision in ss_1 and ss_2, respectively, and create a segment $s'_1 s_2$. Use the inverse of node-split to merge nodes s'_1 and s'_2 into a node s', reducing the number of children of s by one.

In the course of our algorithm, an analogue of the pin-extraction operation extracts a spur from one group into an "adjacent" group. This requires a well-defined adjacency relation between groups. By construction, if a segment uv connects nodes in different clusters, both u and v are leaves or both are root nodes. For every pair of clusters, $C(u)$ and $C(v)$, with adjacent roots, u and v, create a *group* $G_{uv} = C(u) \cup C(v)$; see Figure 24. By construction, the groups are pairwise disjoint. Two groups are called *adjacent* if they have two adjacent leaves in the image graph.

Recall that a maximal path in each cluster is represented by benchmark vertices (leaves and spurs). We denote by $[u_1; \ldots; u_k]$ (using semicolons) a maximal path inside a group defined by the

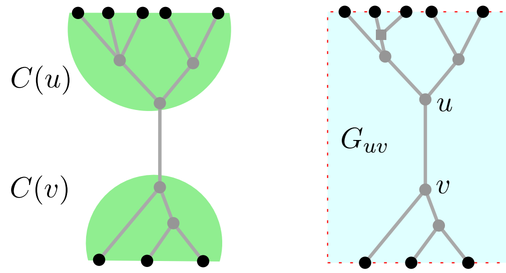


Figure 24: The formation of a group G_{uv}, containing clusters $C(u)$ and $C(v)$. Leaf nodes are shown as black dots.

benchmark vertices $u_1, \ldots, u_k$. For a given group G_{uv}, let $\mathcal{P}$ denote the set of maximal paths with vertices in G_{uv}; and let $\mathcal{B}$ be the set of subpaths in $\mathcal{P}$ between consecutive benchmark vertices.

Remark 2. *By invariants (I1)–(I3), a path in $\mathcal{P}$ of a group G_{uv} has alternating benchmark vertices between $C(u)$ and $C(v)$. Consequently, every path in $\mathcal{B}$ has one endpoint in $C(u)$ and one in $C(v)$, and each spur in G_{uv} is incident to two paths in $\mathcal{B}$.*

Spur-elimination algorithm. Assume that $\mathcal{G}$ is a partition of the nodes of the image graph into groups satisfying (I1)–(I4). We consider one group at a time, and eliminate all spurs from one cluster of that group. When we process one group, we may split it into two groups, create a new group, or create a new spur in an adjacent group (similar to pin-extraction in Section 4). The latter operation implies that we may need to process a group several times. Termination is established by showing that each operation reduces a weighted sum of the number of benchmark vertices (i.e., spurs and boundary vertices). Initially, the number of benchmarks is $O(n)$.

Algorithm spur-elimination$(P, \mathcal{G})$.
While P contains a spur, do:

1. Choose a group $G_{uv} \in \mathcal{G}$ that contains a spur, w.l.o.g. contained in cluster $C(u)$, and create its supporting data structures (described in Section 5.1 below).
2. While $T[u]$ contains an interior node, do:
   (a) If u contains no spurs and is incident to only two edges uv and uw, eliminate u with a merge operation. Rename node w to u which becomes the new root of the tree $T[u]$.
   (b) If u contains spurs, eliminate them as described in Section 5.2.
   (c) If u contains no spurs, split G_{uv} into two groups along a chain of segments that contains uv as described in Section 5.3. Rename a largest resulting group to G_{uv}.

The detailed description of steps 2b and 2c are in Sections 5.2 and 5.3, respectively. We first present supporting data structures in Section 5.1, and then analyze the algorithm in Section 5.4.

5.1 Data structures

In this section, we describe the data structures that we maintain for a group G_{uv}. We start with reviewing and introducing some notation. Consider a group G_{uv} composed of two binary trees $T[u]$ and $T[v]$ rooted at u and v, respectively. Recall that $\mathcal{B}$ denotes the set of benchmark-to-benchmark paths, each with one benchmark in $T[u]$ and one in $T[v]$. In the algorithm spur-elimination, we dynamically maintain the image trees $T[u] \cup T[v]$, and the set of paths $\mathcal{B}$. In each group G_{uv}, we maintain only $O(|\mathcal{B}|)$ nodes that contain benchmark vertices or have degree higher than 2. Dummy nodes of degree two that contain no benchmark vertices are redundant for the combinatorial representation, and will be eliminated with merge operations. However, a polyline formed by a chain of dummy nodes of degree two cannot always be replaced by a straight-line segment (this might introduce unnecessary crossings). By Remark 1, it suffices to maintain the combinatorial embeddings of the trees $T[u]$ and $T[v]$ (i.e., the counterclockwise order of the incident segments around each node).

The partition of a group into two groups is driven by the partition of the paths in $\mathcal{B}$. For a set $\mathcal{B}' \subset \mathcal{B}$ of benchmark-to-benchmark paths, we define a subtree $T(\mathcal{B}')$ induced by $\mathcal{B}'$ as follows. Let $N = N(\mathcal{B}')$ be the set of nodes that contain endpoints of some path in $\mathcal{B}'$. The tree $T(\mathcal{B}')$ is obtained in two steps: take the minimum subtree of $T[u] \cup T[v]$ that contains all nodes in N, and then merge all nodes of degree two that are not in N. In particular, the nodes of $T(\mathcal{B}')$ include N and the lowest common ancestor of any two nodes in $N \cap C(u)$ and in $N \cap C(v)$, respectively. Denote by $\text{lca}(r, s)$ the *lowest common ancestor* of nodes r and s in $T[u]$ (resp., $T[v]$).

Description of data structures. For the image graph of G_{uv}, we maintain the following data structures.

- We store trees $T[u]$ and $T[v]$ each using the dynamic data structure of [8], which supports $O(1)$-time insertion and deletion of leaves, merging interior nodes of degree 2, subdivision of edges, and lowest common ancestor queries.
- Imagine that G_{uv} is inside an axis-aligned rectangle with the leaves of $T[u]$ along the top edge and leaves of $T[v]$ along the bottom edge (see Figure 25(a)). For each tree, we maintain a left-to-right Euler tour in an order-maintenance data structure [3, 19], which supports insertions immediately before or after an existing item, deletions, and precedence queries, each in $O(1)$ amortized time. For any node w, let $w^\flat$ and $w^\sharp$ respectively denote the first and last occurrences of w in the Euler tour. Note that we have $w^\sharp = w^\flat$ for a leaf w. We refer to the elements of the Euler tour as *tokens*. We write $x < y$ to denote that some token x occurs before ("to the left of") another token y in their common Euler tour.
- We also maintain the cyclic list of all leaves of the tree $T[u] \cup T[v]$ (in the order determined by the Euler tour above).

We now describe data structures for $\mathcal{P}$ and $\mathcal{B}$. For every benchmark-to-benchmark path $[s; t] \in \mathcal{B}$, we assume that s is in $T[u]$ and t is in $T[v]$. A path $[s; t]$ is associated with the intervals $[s^\flat, s^\sharp]$ and $[t^\flat, t^\sharp]$. For two consecutive benchmark-to-benchmark paths $[s_1; t; s_2]$, where t is in $T[v]$, we define the interval $I[s_1; t; s_2] = [s_1^\flat, s_2^\flat]$.

- The set of benchmark-to-benchmark paths $[s; t] \in \mathcal{B}$ is stored in four lists, sorted by $s^\flat$, $s^\sharp$, $t^\flat$, and $t^\sharp$, respectively, with ties broken arbitrarily. The sorted lists can be computed in $O(|\mathcal{B}|)$ time by an Eulerian traversal of the tree.
- For each node s of $T[u]$, let $\mathcal{B}_s$ denote the set of paths $[s; t] \in \mathcal{B}$. We store $\mathcal{B}_s$ in two lists, sorted by $t^\flat$ and $t^\sharp$, respectively.

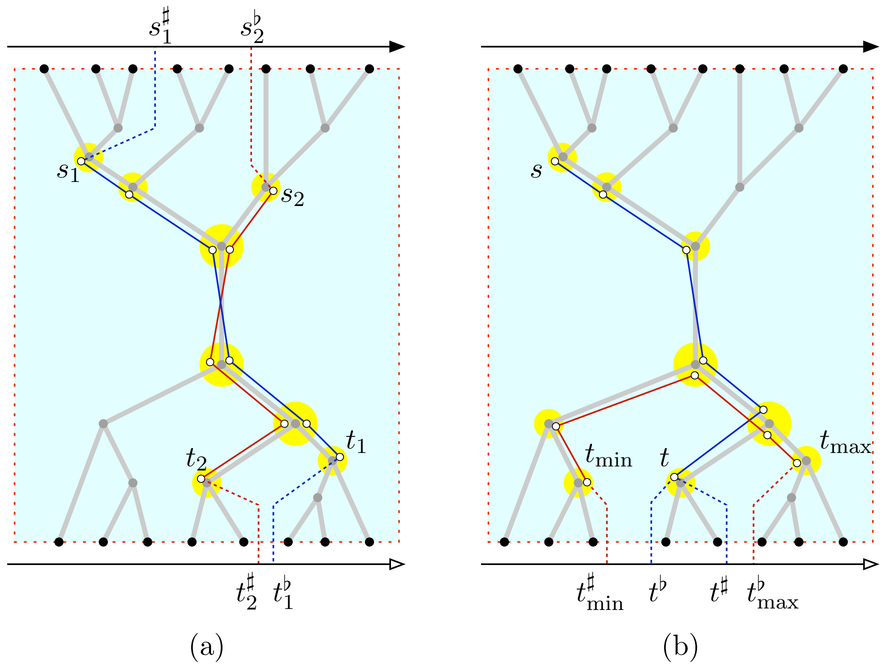


Figure 25: The geometry of crossing benchmark-to-benchmark paths. (a) Paths $[s_1; t_1]$ and $[s_2; t_2]$ cross. (b) If $t_{\min}^\sharp < t^\flat \le t^\sharp < t_{\max}^\flat$, then any benchmark-to-benchmark path $[s; t]$ crosses path $[t_{\min}; t_{\max}]$.

- We use a centered *interval tree* [4] for all $O(n)$ intervals $I[s_1; t; s_2]$ that can report, for a query node q, all intervals containing q in output-sensitive $O(\log n + k)$ time, where k is the number of intervals that contain q. Since the interval endpoints $s^\flat$ are already sorted, the interval tree can be constructed in $O(|\mathcal{B}|)$ time. The interval tree can handle the deletion of an interval in $O(1)$ time (without re-balancing, hence maintaining the $O(\log n + k)$ query time).

All data structures described in this section can be constructed in $O(|\mathcal{B}|)$ preprocessing time.

Crossing paths. The data structure described above can determine in $O(1)$ time whether two paths in $\mathcal{B}$ cross. Straightforward case analysis implies the following characterization of path crossings (refer to Figure 25(a)).

Lemma 10. *Let s_1 and s_2 be arbitrary nodes in tree $T[u]$, and let t_1 and t_2 be arbitrary nodes in $T[v]$. Paths $[s_1; t_1]$ and $[s_2; t_2]$ cross if and only if either (1) $s_1^\sharp < s_2^\flat$ and $t_1^\flat > t_2^\sharp$, or (2) $s_2^\sharp < s_1^\flat$ and $t_2^\flat > t_1^\sharp$.*

5.2 Eliminating spurs from a root

We describe step 2b of Algorithm *spur-elimination*. Suppose that the root node u contains a spur. The following operation eliminates all spurs from u, but the resulting cluster $C(v)$ need not satisfy (I2) and (I3), and we need to perform other operations to restore these properties. Refer to Figure 26(a)–(b) for an example.

spur-shortcut(u). Assume that G_{uv} satisfies invariants (I1)–(I4), and u contains a spur. Replace every path $[t_1; u; t_2]$ by $[t_1; t_2]$. Let $\mathcal{S}$ be the set of all such modified paths.

Lemma 11. spur-shortcut *is ws-equivalent and maintains properties (I1) and (I4).*

Proof. The operation is equivalent to a sequence of spur-reduction operations: First perform spur-reduction(v, u). In a BFS traversal of all nodes x of $T[v]$, except for the root, perform spur-reduction$(x, \text{parent}(x))$. All these operations satisfy spur-reduction's constraints. Initially, every path through the node x has an edge in the segment $x\,\text{parent}(x)$, by (I2). The BFS traversal ensures that this property still holds when the algorithm performs spur-reduction$(x, \text{parent}(x))$. ☐

Note that for every path $[t_1; u; t_2]$, both t_1 and t_2 are in $T[v]$ (cf. Remark 2) and path $[t_1; t_2]$ is uniquely defined by (I1). However, a maximal path in $C(v)$ that contains $[t_1; t_2]$ violates (I2), and if $t_1 = t_2$ is a leaf in $C(v)$, then it forms a spur that may violate (I3). We proceed with a sequence of "repair" steps to restore them, after which the total number of benchmark vertices decreases by at least $|\mathcal{S}|$. The following three steps restore (I2) and (I3) when t_1 and t_2 are in ancestor-descendent relation, that is, $\text{lca}(t_1, t_2) \in \{t_1, t_2\}$. Let $\min(t_1, t_2)$ denote the node in $\{t_1, t_2\}$ farther from the root.

For every path $[t_1; t_2] \in \mathcal{S}$, do

1. If $\text{lca}(t_1, t_2) \in \{t_1, t_2\}$ and $t_1 \neq t_2$, then replace $[t_1; t_2]$ with $[\min(t_1, t_2)]$.
2. If $t_1 = t_2$ and t_1 is not a leaf of $T[v]$ that has degree two in the image graph, then replace $[t_1; t_2]$ with $[t_1]$.
3. If $t_1 = t_2$ and t_1 is a leaf of $T[v]$ that has degree two in the image graph, then do: by (I3), node t_1 is adjacent to a unique node $z \notin G_{uv}$ and z is incident to a single segment in the cluster containing z. Subdivide such segment creating a new node z^* (added to the cluster containing z), and replace every path $[z, t_1, z]$ with $[z^*]$. See Figure 26(b)–(c) for an example.

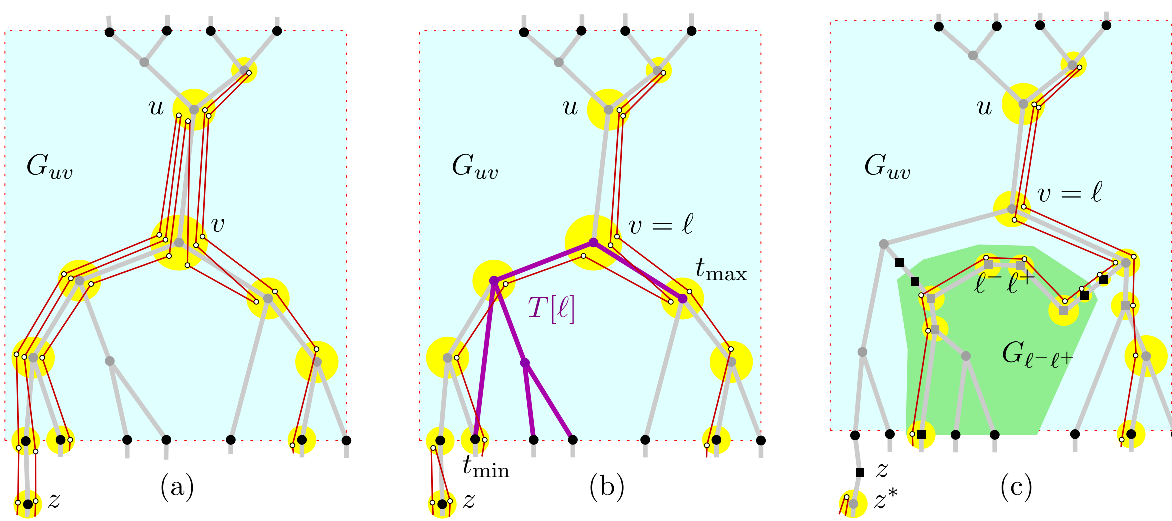


Figure 26: (a) Node u contains spurs. (b) After eliminating spurs, $T[v]$ does not satisfy (I2). (c) The analogues of pin-extraction and V-shortcut.

These steps restore (I3) at all leaves, and (I2) for the affected paths $[t_1; t_2] \in \mathcal{S}$. Note that these steps are ws-equivalent: Steps 1–2 do not modify the polygon (they change only the benchmarks); and step 3 is analogous to pin-extraction(t_1, z).

We are left with paths $[t_1; t_2] \in \mathcal{S}$ where t_1 and t_2 are in different branches of $T[v]$. In this case, we perform an elaborate version of the V-shortcut operation, that creates a new group. For every node ℓ of $T[v]$, let $\mathcal{S}_\ell$ be the set of paths $[t_1; t_2] \in \mathcal{S}$ such that $\text{lca}(t_1, t_2) = \ell$. Consider every node ℓ of $T[v]$ where $\mathcal{S}_\ell \neq \emptyset$ in a bottom-up traversal of $T[v]$; and create a new group $G_{\ell^-\ell^+}$ as follows (refer to Figure 26).

Let N^- (resp., N^+) be the set of nodes t_1 (resp., t_2) such that there is a path $[t_1; t_2] \in \mathcal{S}_\ell$, and t_1 is in the left (resp., right) subtree of ℓ. Let $N = N^- \cup N^+$. Sort the nodes $t_1 \in N^-$ by $t_1^\sharp$, and let $t_{\min}$ be the minimum node; and similarly sort the nodes $t_2 \in N^+$ by $t_2^\flat$, and let $t_{\max}$ be the maximum node. The following lemma shows that interior nodes of the path from $t_{\min}$ to ℓ in $T[v]$ have no right branches, and the interior nodes of the path from $t_{\max}$ to ℓ have no left branches.

Lemma 12. *If there is a path $[s; t] \in \mathcal{B} \setminus \mathcal{S}_\ell$ such that $t_{\min}^\sharp < t^\flat \leq t^\sharp < t_{\max}^\flat$, then it crosses some path in $\mathcal{S}_\ell$, hence P is not weakly simple.*

Proof. Let C be the path between $t_{\min}$ and $t_{\max}$ in $T[v]$. Refer to Figure 25(b). By the choice of ℓ (in a bottom-up traversal of $T[v]$), we have $\mathcal{S}_{\ell'} = \emptyset$ for all descendants of ℓ. Path $[s; t]$ reaches C at some interior node $t^* \in C$, and then continues to ℓ, and farther to parent(ℓ). If t^* is in a left (resp., right) subtree of ℓ, then $[s; t]$ crosses every path in $\mathcal{S}_\ell$ that starts at $t_{\min}$ (resp., ends at $t_{\max}$). ☐

We can find the set N' of nodes t such that $t_{\min}^\sharp < t^\flat \leq t^\sharp < t_{\max}^\flat$, in $O(|N'| + \log n)$ time, by binary search in the list of leaves to find all the leaves between $t_{\min}$ and $t_{\max}$, and by lowest common ancestor queries to find nodes in N'. The algorithm reports that the input polygon is not weakly simple and halts if some node in N' has a path satisfying the condition in Lemma 12. We can now assume that $N' \subset N$. The nodes in N induce a binary tree, denoted $T[\ell]$, of size at most $2|N|$: its nodes are all nodes in N and the lowest common ancestors of consecutive nodes in N^- and N^+ respectively. Note that a segment of $T[\ell]$ might not correspond to a segment of $T[v]$ (see Figure 26(b)). Denote by C^* the path between $t_{\min}$ and $t_{\max}$ in $T[\ell]$.

We now define the changes in the image graph. Every node $t \in N \setminus C^*$ is deleted from G_{uv}, and added to the new group. Create two nodes, ℓ^- and ℓ^+, in $G_{\ell^-\ell^+}$ sufficiently close to ℓ in the wedge between the two children of ℓ, and connect them by a segment $\ell^-\ell^+$. Duplicate each node $t \in C^* \setminus \{\ell\}$, by creating a node t' (added to $G_{\ell^-\ell^+}$) sufficiently close to t, and add a segment tt'. Subdivide every segment tt' with two new boundary nodes, t_{leaf} (added to $T[v]$) and t'_{leaf} (added to $G_{\ell^-\ell^+}$). The nodes t or t' might now have degree 4. Adjust the image graph so that the group trees are binary. Finally partition the nodes in $G_{\ell^-\ell^+}$ into two trees, $T[\ell^-]$ and $T[\ell^+]$, rooted at ℓ^- and ℓ^+, respectively.

We now define the changes in the polygon. Replace every path $[t; t_1] \in \mathcal{S}_\ell$, where $t \in C^*$, by $[t'; t_1]$ if it is adjacent to a path $[t; t_2] \in \mathcal{S}_\ell$, i.e., replacing the path $[t_1; t; t_2]$ by $[t_1; t'; t_2]$. Otherwise, replace $[t; t_1]$ by $[t_{\text{leaf}}; t'_{\text{leaf}}; t_1]$. Now we can build $\mathcal{B}'$ as the set of benchmark-to-benchmark paths $[t'_1; t'_2]$ where $t'_1, t'_2 \in G_{\ell^-\ell^+}$ in $O(|\mathcal{B}'|)$ time.

To prove ws-equivalence, we consider the changes in the polygon. These amount to a sequence of ws-equivalent primitives: a node-split at ℓ, a sequence of node-splits along the chain C from ℓ to $t_{\min}$ and $t_{\max}$, respectively, subdivision operations that create the new leaf nodes, and merge operations at degree two nodes that no longer contain spurs. The creation of new groups takes $O(|\mathcal{S}_\ell| + \log n)$ time and $O(|\mathcal{S}_\ell|)$ paths in $\mathcal{B}$ are removed or modified in G_{uv}. Thus the data structures for G_{uv} are updated in $O(|\mathcal{S}_\ell| \log n)$ time. Overall, operation spur-shortcut(u) and the repair steps that follow take $O(|\mathcal{S}| \log n)$ time.

5.3 Splitting a group in two

In this section we describe step 2c of Algorithm spur-elimination$(P, \mathcal{G})$. Assume that G_{uv} satisfies invariants (I1)–(I4) and u contains no spur.

Denote the left and right child of u by u^- and u^+, respectively. Let $\mathcal{B}^-, \mathcal{B}^+ \subset \mathcal{B}$, resp., be the set of benchmark-to-benchmark paths that contain u^- and u^+. We split G_{uv} into two groups induced by $\mathcal{B}^-$ and $\mathcal{B}^+$, respectively. Refer to Figure 27.

It would be easy to compute the groups induced by $\mathcal{B}^-$ and $\mathcal{B}^+$ in $O(|\mathcal{B}|)$ time. However, for an overall $O(n \log n)$-time algorithm, we can afford $O(\min(|\mathcal{B}^-|, |\mathcal{B}^+|))$ time for the split operation, and an additional $O(\log n)$ time for each eliminated spur and each node that we split into two nonempty nodes. Without loss of generality, we may assume $|\mathcal{B}^-| \leq |\mathcal{B}^+|$. The group induced by $|\mathcal{B}^-|$ can be computed from scratch in $O(|\mathcal{B}^-|)$ time, and we construct the group for $\mathcal{B}^+$ by modifying G_{uv}, and updating the corresponding data structures.

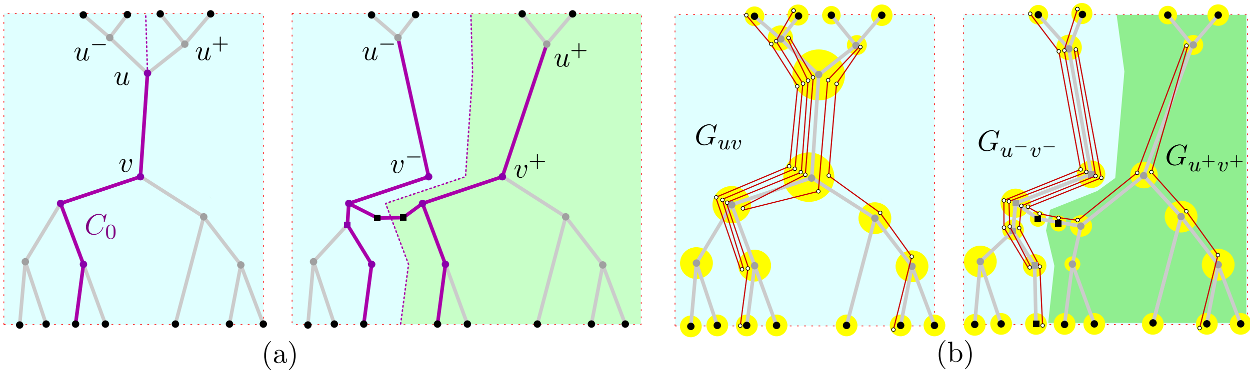


Figure 27: Splitting group G_{uv}. (a) Changes in the image graph. (b) Changes in the polygon.

First, we find $\mathcal{B}^-$ and $\mathcal{B}^+$. Compute $\mathcal{B}^-$ using the list of paths $[s;t] \in \mathcal{B}$ sorted by $s^\sharp$ or $s^\flat$. Since both lists naturally split into corresponding lists for $\mathcal{B}^-$ and $\mathcal{B}^+$, we can split these lists in $O(\min(|\mathcal{B}^-|, |\mathcal{B}^+|)) = O(|\mathcal{B}^-|)$ time. To construct the list of $\mathcal{B}^+$ sorted by $t^\sharp$ and $t^\flat$, we start with the corresponding lists for $\mathcal{B}$, and delete all elements of $\mathcal{B}^-$ in $O(|\mathcal{B}^-|)$ time. To compute the lists sorted by $t^\sharp$ and $t^\flat$ for $\mathcal{B}^-$, we shall first compute the subtree $T[v^-]$ induced by $\mathcal{B}^-$. However, we can already find the maximum $t^\sharp$ of a path $[s;t] \in \mathcal{B}^-$ in $O(|\mathcal{B}^-|)$ time.

Next, we test for crossings between the paths in $\mathcal{B}^-$ and the paths in $\mathcal{B}^+$. Let $t^\sharp_-$ be the maximum $t^\sharp$ of a path $[s;t] \in \mathcal{B}^-$, and $t^\flat_+$ the minimum $t^\flat$ of a path $[s;t] \in \mathcal{B}^+$. By Lemma 10, there is such a crossing if and only if $t^\flat_+ < t^\sharp_-$, which can be determined in $O(1)$ time using our order-maintenance structures. If a crossing is detected, the algorithm reports that P is not weakly simple and halts.

Trees $T[u^-]$ and $T[u^+]$ are simple subtrees of $T[u]$; but splitting $T[v]$ is nontrivial. We use binary search in the Eulerian cycle of all leaves to find the rightmost leaf ℓ_0 in $T[v]$ such that $\mathcal{B}_{\ell_0} \cap \mathcal{B}^- \neq \emptyset$, if such a leaf exists, otherwise the leftmost leaf ℓ_0 in $T[v]$. Let $C_0 = [\ell_0; u]$. We do not compute the path C_0 explicitly, as it may contain more than $O(|\mathcal{B}^-|)$ nodes, but we can test whether a query node t of $T[v]$ is in C_0 in $O(1)$ time by checking whether $\mathrm{lca}(\ell_0, t) = t$. Since the paths in $\mathcal{B}^-$ and $\mathcal{B}^+$ do not cross, all nodes of $T[v^-]$ are in or to the left of the chain C_0, and all common nodes of $T[v^-]$ and $T[v^+]$ are in C_0. The image graph of $T[v^-]$ can be computed from scratch using $\mathcal{B}^-$ in $O(\mathcal{B}^-)$ time. Replace each node t of $T[v^-]$ that is in C_0 by a duplicate copy t^- located sufficiently close to t, to the right of t. The tree $T[v^+]$ is computed from $T[v]$ by node deletion and merge operations as follows. First delete all nodes that are in $T[v^-]$ but not in C_0. For every node t of $T[v^-]$ that lies in C_0, if t has degree two in $T[v^-]$ and $\mathcal{B}_t^+ = \emptyset$, then it would be a degree two node in $T[v^+]$ with no spurs, and so we can delete t by merging its two incident segments. Let v^+ be the node not in $T[u^+]$ adjacent to u^+. The resulting $T[v^+]$ becomes a tree induced by $\mathcal{B}^+$. It remains to resolve the connections between trees.

Let $\mathcal{V}^0$ denote the set of chains $[s_1; t; s_2]$ such that $[s_1; t] \in \mathcal{B}^-$ and $[t; s_2] \in \mathcal{B}^+$. The spurs at t on all chains $[s_1; t; s_2] \in \mathcal{V}^0$ will be eliminated (they will become adjacent leaves in the two resulting groups). $\mathcal{V}^0$ can be found with a query for u in the interval tree. Let N^0 be the set of all nodes t such that $[s_1; t; s_2] \in \mathcal{V}^0$. Each node $t \in N^0$ is in C_0 and, therefore, has a copy t^- in $T[v^-]$. Create a segment between t and t^-, and subdivide the segment t^-t with two new nodes t^-_{leaf} and t_{leaf} in $T[v^-]$ and $T[v^+]$, respectively. The degree of nodes t or t^- might increase to 4; and so we adjust the image graphs so that both trees are binary. The image graph is now split into groups $G_{u^-v^-}$ and $G_{u^+v^+}$.

We next define the changes in the polygon. Replace every chain $[s_1; t; s_2] \in \mathcal{V}^0$ with a new chain $[s_1; t^-_{\text{leaf}}; t_{\text{leaf}}; s_2]$, while also replacing the corresponding paths in the lists $\mathcal{B}^-$ and $\mathcal{B}^+$ in $O(|\mathcal{V}^0|)$ time. In the sorted lists for $\mathcal{B}^-$ and $\mathcal{B}^+$, this is done by deletions and reinsertions. Note that all leaves t^-_{leaf} (resp., t_{leaf}) are at the end (resp., beginning) of the Euler tour of $T[v^-]$ (resp., $T[v^+]$), so deletions can be performed in $O(|\mathcal{V}^0|)$ time; and insertions take $O(|\mathcal{V}^0| \log n)$ time.

The changes in the polygon are equivalent to a sequence of ws-equivalent primitives: a node-split operation at u, followed by a sequence of node-splits along the chain C_0 from ℓ_0 to u, and subdivision operations that create the new leaf nodes between the two groups. The interval tree is updated by deleting the intervals that contain u, and the query time remains the same output-sensitive $O(\log n + k)$. Consequently, we can split G_{uv} in $O(\min(|\mathcal{B}^-|, |\mathcal{B}^+|) + |\mathcal{V}^0| \log n + \log n)$ time.

5.4 Analysis of the spur-elimination algorithm

Lemma 13. *Given m benchmark vertices,* spur-elimination$(P, \mathcal{G})$ *takes $O(m \log m)$ time.*

Proof. Let σ be the number of spurs, β the number of benchmark vertices at the leaves of clusters, and let $\phi = 2\sigma + \beta$. Initially, $\phi = O(m)$ by (I1). All operations in spur-elimination monotonically decrease both σ and ϕ. Step 2b decreases ϕ by the number of spurs at u, and steps 2a and 2c both maintain ϕ. In particular, Step 2c converts some spurs into pairs of adjacent benchmark vertices at leaves. Consequently, the number of benchmark vertices remains $O(m)$ throughout the algorithm.

Step 1 creates data structures for new groups: For a group containing m benchmarks, all supporting data structures can be computed in $O(m)$ time, that is, in $O(1)$ time per benchmark. A new benchmark v appears in a group when (i) a benchmark is extracted into an adjacent group, or (ii) a group of size m is split and v is part of the smaller group of size at most $m/2$. Extraction strictly decreases ϕ, so it occurs $O(m)$ times. The total number of benchmarks that are either present initially or created by extraction is $O(m)$. Each of these benchmarks can move into a group of half-size $O(\log m)$ times. Consequently, there are $O(m \log m)$ new benchmarks overall, and the time spent on all instances of Steps 1 is $O(m \log m)$.

Step 2a removes an interior node of degree two; the update of supporting data structures takes

$O(\log m)$ time. Interior nodes are created only when they contain a spur, so at most $O(m)$ interior nodes are ever created, and all instances of Step 2a take $O(m \log m)$ time. Step 2b eliminates $|\mathcal{S}|$ spurs in $O(|\mathcal{S}| \log m)$ time. Eventually, all spurs are eliminated, thus all instances of Step 2b take $O(m \log m)$ time. Step 2c takes $O(\min(|\mathcal{B}^-|, |\mathcal{B}^+|) + |\mathcal{V}^0| \log m + \log m)$ time. By a standard heavy-path decomposition argument, the terms $\min(|\mathcal{B}^-|, |\mathcal{B}^+|)$ contribute $O(m \log m)$ time. Every chain in $\mathcal{V}^0$ corresponds to a spur that is destroyed in a step 2c (and no new spurs are created in step 2c), therefore the terms $O(|\mathcal{V}^0| \log m)$ sum to $O(m \log m)$ over the course of the algorithm. Since every execution of step 2c increases the number of groups by one, and this step is repeated $O(m)$ times, the $\log m$ terms sum to $O(m \log m)$ in the entire algorithm. ☐

Algorithm spur-elimination$(P, \mathcal{G})$ returns a polygon P', a set $\mathcal{G}'$ of groups, and a set $\mathcal{B}'$ of benchmark-to-benchmark paths, each of which connects two leaves in two different clusters of a group. In each group G_{uv}, the trees $T[u]$ and $T[v]$ have no interior nodes, thus G_{uv} consists of two single-node clusters $C(u) = \{u\}$ and $C(v) = \{v\}$, connected by a single edge uv. Consequently, the image graph is 2-regular. We can now decide whether P' is weakly simple in $O(n)$ time similarly to [6, Section 3.3]. The polygon P' is weakly simple if and only if the image graph is connected and each group contains precisely one benchmark-to-benchmark path. These properties can be verified by a simple traversal of the image graph and P' in $O(n)$ time. This completes the proof of Theorem 1.

6 Perturbing weakly simple polygons into simple polygons

In Sections 3–5, we have presented an algorithm that decides, in $O(n \log n)$ time, whether a given n-gon P is weakly simple. If P is weakly simple, then for every $\varepsilon > 0$ it can be perturbed into a simple polygon by moving each vertex a distance at most ε. In this section we show how to find, for any $\varepsilon > 0$, a simple polygon Q with $2n$ vertices such that $\mathrm{dist}_F(P, Q) < \varepsilon$. Let P' and P'' be the polygons obtained after the bar-simplification and spur-elimination phases of the algorithm, respectively. P'' has $O(n)$ vertices, none of which is a fork or a spur. Using the results in [6, Section 3], we can construct a simple polygon $Q'' \in \Phi(P'')$ in $O(n)$ time. In this section, we show that we can reverse the sequence of operations in $O(n \log n)$ time and perturb P as well into a simple polygon $Q \in \Phi(P)$.

Combinatorial representation by bar-signatures. A perturbation of a weakly simple polygon has a combinatorial representation, called a signature, which consists of total orders of the overlapping edges in all segments of the image graph (cf. Section 2). In the absence of forks, every edge lies in a segment, and the size of such a signature is $O(n)$. However, the signature may have size $\Theta(n^2)$ in the presence of forks. When our algorithm eliminates forks from a polygon, it may create $\Theta(n^2)$ dummy vertices and edges, which would again lead to a signature of size $\Theta(n^2)$. For reversing the operations of the algorithm in Sections 3–5, we introduce a new combinatorial representation of size $O(n)$ that maintains the total order of the edges in each bar that are outside of clusters.

For $n \geq 3$, let $P = (p_0, \ldots, p_{n-1})$ be a weakly simple polygon with image graph G. Assume that the sober nodes of G are partitioned into a set $\mathcal{C}$ of disjoint clusters satisfying invariants (I1)–(I4) such that every bar is either entirely in a cluster or outside of all clusters. Let $Q = (p'_0, \ldots, p'_{n-1})$ be a simple polygon such that $|p_i, p'_i| < \varepsilon_0 = \varepsilon_0(P)$ for all $i = 0, \ldots, n-1$. We may assume that G has no vertical segments (so that the above-below relationship is defined between disjoint segments parallel to a bar). In each segment uv of G outside of clusters, the above-below relationship yields a total ordering over the edges of Q that contain uv. For each bar b outside of clusters, the total orders of the segments along b are consistent (since the above-below relationship between two edges is the same in every corridor). Consequently, the transitive closure of these total orders is a partial order over all edges in b. Consider a linear extension of such a partial order. The collection of these total orders for all bars in P is a *bar-signature* of Q. Since the linear extensions need not be unique, a polygon $Q \in \Phi(P)$ may have several bar-signatures.

Given a bar-signature of a perturbation of P, we can (re)construct an approximate simple polygon Q' as follows; refer to Figure 28. For every bar $b = uv$ of G outside of clusters, let the *volume* $\mathrm{vol}(uv)$ be the number of edges of P that lie on b. Place $\mathrm{vol}(uv)$ parallel line segments, called *lanes*, between ∂D_u and ∂D_v in the region U_ε, ordered from bottom to top (the lanes contain the edges of Q'). For the i-th edge pq in the total order of b, let the corresponding edge in Q' be the shortest edge connecting ∂D_p and ∂D_q in the i-th lane. For each cluster $C(u)$, denote by $R(u)$ the union of all disks D_v, $v \in C(u)$, and all corridors between nodes in $C(u)$. If $C(u)$ contains only the node u, then $R(u) = D_u$, but $R(u)$ is always simply connected since $C(u)$ induces a tree $T[u]$. For each cluster $C(u)$, construct a noncrossing polyline matching, between the endpoints of the edges in $\partial R(u)$, that connects the endpoints corresponding to a maximal subpath in $T[u]$. The edges in the lanes and the perfect matchings in the regions $R(u)$ produce a polygon Q'. If the Euclidean diameter of each region $R(u)$ is at most δ, then the Fréchet distance between P and Q' is at most $\varepsilon + \delta$. Denote by $\Psi(P)$ the set of all simple polygons that can be constructed in this manner from a bar-signature for some ε, $0 < \varepsilon < \varepsilon_0$.

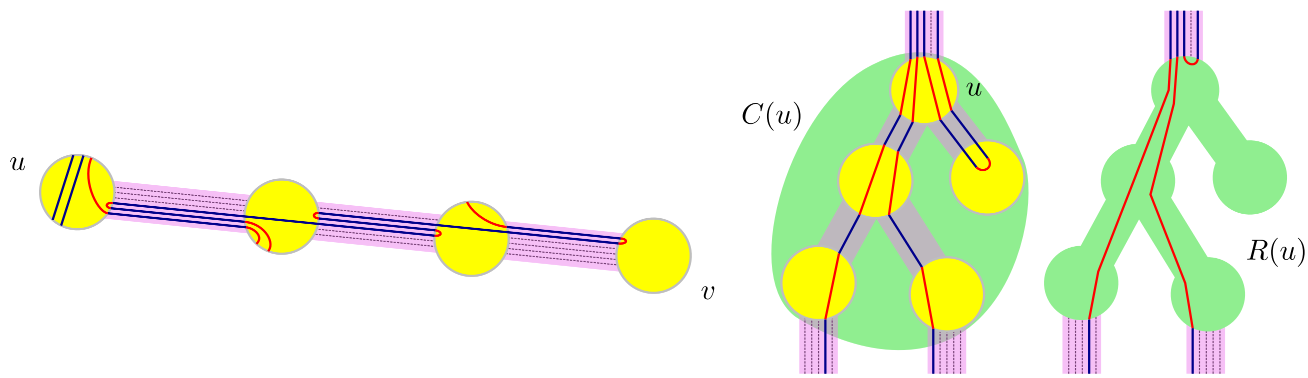


Figure 28: Construction of a simple polygon $Q' \in \Psi(P)$ from a bar-signature. Left: Bar uv of a simple polygon obtained from an order compatible with the polygon shown in Figure 2(c). Right: maximal paths of Q and Q' inside clusters.

Spur elimination. If a given n-gon is weakly simple, our decision algorithm computes a polygon P'', which is ws-equivalent to P and represented implicitly by a cyclic sequence of benchmark nodes. Specifically, P'' is represented by an image graph G'', a set $\mathcal{G}''$ of groups, a set $\mathcal{B}''$ of benchmark-to-benchmark paths, and for every group $G_{uv} \in \mathcal{G}''$, a linear order of the paths in $\mathcal{B}''$ that cross the corridor N_{uv} between D_u and D_v. Consequently, the decision algorithm provides a bar-signature for the weakly simple polygon P''.

We show that, by reversing the steps of Algorithm $\mathsf{spur\text{-}elimination}(P', \mathcal{G}')$, we can compute a bar-signature of P' in $O(n \log n)$ time. If a group G_{uv} has been split in some step 2c (cf. Section 5.3), we can construct an ordering of the benchmark-to-benchmark paths of G_{uv} by concatenating the orders of $\mathcal{B}^-$ and $\mathcal{B}^+$ (the sets of benchmark-to-benchmark paths of the resulting two groups).

If G_{uv} had spurs eliminated from u in some step 2b (cf. Section 5.2), we reverse each of the steps in the following manner. Recall that if a new group $G_{\ell-\ell+}$ was created, then every path $[t'_1; t'_2] \in \mathcal{B}'$ in that group was created from a concatenation of two paths $[t_1; u]$ and $[t_2; u]$. Use the ordering of the paths in $\mathcal{B}'$ to insert the paths $[t_1; u]$ and $[t_2; u]$ into the ordering of $\mathcal{B}$ so that they form nested spurs, i.e., if $[t'_1; t'_2]$ is the topmost edge in $\mathcal{B}'$, $[t_1; u]$ (resp. $[t_2; u]$) should be the leftmost (resp., rightmost) path (without loss of generality, we use the orientation of Figure 26). Identify the leftmost path in the segment that connects ℓ and its right child and place all nested paths that created $G_{\ell-\ell+}$ immediately to its left.

If one or more spurs were created at a node z in an adjacent group, we can find the position of the edges incident to each spur in the ordering of the adjacent group. Using this order, we can identify the first path in G_{uv} to the right of the edges incident to $[z]$. Then, immediately to the left of such a path, we can place the paths $[t_1; u; t_1]$ that generated the spurs at z. The relative order of these paths is the same as the one obtained by reversing a spur-reduction, described in the proof of Lemma 4, and therefore produces a simple polygon. If a path $[t_1; u; t_2]$ is simplified to $[t_1]$ (Step 1 with $\min(t_1, t_2) = t_1$ without loss of generality; or Step 2), we can proceed analogously to the reversal of a crimp-reduction (cf. Lemma 1) from a path $[t_1; u; t_2; s]$ to $[t_1; s]$. Identify the path $[t_1; s]$ in the ordering of $\mathcal{B}$ and replace it with the paths $[t_1; u]$, $[u; t_2]$, and $[t_2; s]$ in this order.

Bar simplification. The bar-signature determines all segments between adjacent clusters. Using these orders, we can reverse the operation pin-extraction(u, v) assigning the same order for the edges in uv as the order of its adjacent benchmark-to-benchmark paths. V-shortcut is also trivially reversible by concatenating the order of segments that get merged.

Updating the bar-signature when we reverse an L-shortcut operation is a bit more challenging. Determining the edge order in segments vw and vu_1 can be trivially done by just concatenating the order of merged segments. But phase (1) introduces a crimp in some cross-chains, and the reverse operation, crimp-reduction, may require nontrivial reordering in the bar-signature. Suppose that P' is obtained from P after a crimp-reduction. The proof of Lemma 1 shows a straightforward way to obtain a bar-signature of a polygon in $\Psi(P)$ given a polygon in $\Psi(P')$. However, obtaining a bar-signature of $Q' \in \Psi(P')$ given $Q \in \Psi(P)$ requires identifying W_{top} and W_{bot}, which takes $O(n)$ time.

In order to handle the reversal of phase (1) in $O(1)$ time, we divide the signature of each bar into pieces. Recall that the bar-simplification algorithm does not eliminate any cross-chains from D_b, and when bar-simplification terminates, only one-edge cross-chains remain in the interior of D_b. Let K denote the set of cross-chains of D_b. The segments of the image graph that cross the ellipse D_b, and the bar-signatures of these edges yield a linear order (from left to right) of K; and the cross-chains subdivide D_b into $|K| + 1$ regions. We maintain a linear order for the edges along the bar in each such region (including the boundary of the region), and denote the set of these edges in b by $E_1, \ldots, E_{|K|+1}$.

We reverse phases (2) and (3) of L-shortcut(v, TR) as follows (applying reflections for other L-shortcut operations if necessary). Assign the new edges $[u_1, u_2]$ the highest lanes in the ordering of the appropriate E_i, maintaining the relative order of affected paths. To reverse phase (1), first notice that the three edges in the crimp $[u_1, u_2, u_1, u_2]$ are part of a cross-chain, consequently they appear in two consecutive subsets E_i and E_{i+1}. In the ordering of the left (resp., right) subset, assign the new edge $[u_1, u_2]$ to the highest (resp., lowest) position among the positions of the three edges $[u_1, u_2]$.

When all operations in the bar simplification algorithm have been reversed, we have to combine the linear orders of $E_1, \ldots, E_{|K|+1}$ into a total order, a common linear extension of these orders. The intersection of two edge sets, $E_i \cap E_j$ with $i < j$, is either disjoint or contains the edges of the i-th cross-chain. The above-below relationship between the edges of each cross-chain is uniquely determined by Lemma 2, and must be the same in each total order. Therefore, the union of the total orders is a partial order for all edges in the bar. Since the ordering of each subset guarantees that its paths can be realized without crossing, any linear extension of this partial order produces a bar-signature of a simple polygon.

Preprocessing. The cluster formation and new-bar-expansion consist of subdivision operations that do not influence the order of edges that define the bar-signature. If an edge $[v, w]$ in a bar b is subdivided into $[v, v', w]$, where $[v', w]$ is in D_b, we can assign $[v, w]$ to the same lane of $[v', w]$ in the ordering of edges in b. The crimp-reduction operations can be reversed by making the three edges that form a new crimp consecutive in the ordering, as in the proof of Lemma 1.

We have shown how to maintain bar-signatures while reversing the operations of our algorithms, in time proportional to those operations. For every $\varepsilon > 0$, the bar-signatures yield a perturbation of a weakly simple polygon P into a simple polygon $Q \in \Phi(P)$ with $2n$ vertices, where each vertex $[u]$ of P corresponds to two vertices of Q on the circle ∂D_u. This completes the proof of Theorem 2.

7 Conclusion

We presented an $O(n \log n)$-time algorithm for deciding whether a polygon with n vertices is weakly simple. Weak simplicity of polygons has a natural generalization for planar graphs [6, Appendix D]. We can define the *weak embedding* for graphs in terms of Fréchet distance. A graph $H = (V, E)$ can be considered a 1-dimensional simplicial complex. A *drawing* of H is a continuous map of H to $\mathbb{R}^2$. The Fréchet distance between two drawings, P and Q, of H is defined as $\mathrm{dist}_F(P, Q) = \inf_{\phi: H \to H} \max_{x \in H} \mathrm{dist}(P(\phi(x)), Q(x))$, where ϕ is an automorphism of H (a homeomorphism from H to itself). Very recently, Fulek and Kynčl [13] gave a polynomial-time algorithm for deciding whether a given drawing of a graph H is weakly simple, i.e., whether a straight-line drawing P of H is within ε Fréchet distance from some embedding Q of H, for all $\varepsilon > 0$. Earlier, efficient algorithms were known only in special cases: when the embedding is restricted to a given isotopy class (i.e., given combinatorial embedding) [12]; and when all n vertices are collinear and the isotopy class is given [1].

We can also generalize the problem to higher dimensions. A polyhedron can be described as a map $\gamma : M \to \mathbb{R}^3$, where M is a 2-manifold without boundary. A simple polyhedron is an injective function. A polyhedron P is weakly simple if there exists a simple polyhedron within ε Fréchet distance from P for all $\varepsilon > 0$. This problem can be reduced to origami flat foldability. The results of [5] imply that, given a convex polygon P and a piecewise isometric function $f : P \to \mathbb{R}^2$ (called *crease pattern*), it is NP-hard to decide if there exists an injective embedding of P in three dimensions $\lambda : P \to \mathbb{R}^3$ within ε Fréchet distance from f for all $\varepsilon > 0$, i.e., if f is *flat foldable*. Given P and f, we can construct a continuous function $g : \mathbb{S}^2 \to P$ mapping each hemisphere of $\mathbb{S}^2$ to P (for a point $x \in P$, the inverse image $g^{-1}(x)$ is a set of two points in opposite hemispheres of $\mathbb{S}^2$). Then, the polyhedron $\gamma = g \circ f$ is weakly simple if and only if f is flat foldable. Therefore, it is also NP-hard to decide whether a polyhedron is weakly simple.

Finally it is an open problem to find a linear-time algorithm for recognizing weakly simple polygons. Chang et al. [6] conjectured that this is possible in the absence of spurs and forks.

Acknowledgements. Research by Akitaya, Aloupis, and Tóth was supported in part by the NSF awards CCF-1422311 and CCF-1423615. Akitaya was supported by the Science Without Borders program. Research by Erickson was supported in part by the NSF award CCF-1408763. We thank Anika Rounds and Diane Souvaine for many helpful conversations that contributed to the completion of this project. We thank the anonymous referees for many useful comments and suggestions.

References

[1] Zachary Abel, Erik D. Demaine, Martin L. Demaine, David Eppstein, Anna Lubiw, and Ryuhei Uehara, Flat foldings of plane graphs with prescribed angles and edge lengths, in *Proc. 22nd Symposium on Graph Drawing*, LNCS 8871, Springer, 2014, pp. 272–283.

[2] Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Martin L. Demaine, Joseph S.B. Mitchell, Saurabh Sethia, and Steven S. Skiena, When can you fold a map?, *Computational Geometry: Theory and Applications* **29** (2004), 23–46.

[3] Michael A. Bender, Richard Cole, Erik D. Demaine, Martin Farach-Colton, and Jack Zito. Two simplified algorithms for maintaining order in a list, *Proc. 10th Annual European Symposium on Algorithms*, LNCS 2461, Springer, 2002, pp. 152–164.

[4] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars, *Computational Geometry: Algorithms and Applications*, third edition, Springer, Berlin, 2008.

[5] Marshall Bern and Barry Hayes, The complexity of flat origami, in *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 1996, pp. 175–183.

[6] Hsien-Chih Chang, Jeff Erickson, and Chao Xu, Detecting weakly simple polygons, in *Proc. 26th ACM-SIAM Symposium on Discrete Algorithm*, SIAM, 2015, pp. 1655–1670.

[7] Bernard Chazelle, Triangulating a simple polygon in linear time, *Discrete & Computational Geometry* **6** (1991), 485–524.

[8] Richard Cole and Ramesh Hariharan, Dynamic LCA queries on trees, *SIAM J. Comput.* **34(4)** (2005), 894–923.

[9] Robert Connelly, Erik D. Demaine, and Günter Rote, Infinitesimally locked self-touching linkages with applications to locked trees, in *Physical Knots: Knotting, Linking, and Folding of Geometric Objects in* $\mathbb{R}^3$, American Mathematical Society, Povidence, RI, 2002, pages 287–311.

[10] Pier Francesco Cortese, Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia, On embedding a cycle in a plane graph, *Discrete Mathematics* **309(7)** (2009), 1856–1869.

[11] Andrea Francke and Csaba D. Tóth, A census of plane graphs with polyline edges, *SIAM J. Discrete Math.* **31(2)** (2017), 1174–1195.

[12] Radoslav Fulek, Embedding graphs into embedded graphs, preprint, `arXiv:1608.02087`, 2016.

[13] Radoslav Fulek and Jan Kynčl, Hanani-Tutte for approximating maps of graphs, preprint, `arXiv:1705.05243`, 2017.

[14] Branko Grünbaum, Polygons: Meister was right and Poinsot was wrong but prevailed, *Beiträge zur Algebra und Geometrie* **53(1)** (2012), 57–71.

[15] Piotr Minc, Embedding of simplicial arcs into the plane, *Topology Proceedings* **22** (1997), 305–340.

[16] Ares Ribó Mor, *Realization and Counting Problems for Planar Structures: Trees and Linkages, Polytopes and Polyominoes*, Ph.D. thesis, Freie Universität Berlin, 2006.

[17] Michael Ian Shamos and Dan Hoey, Geometric intersection problems, in *Proc. 17th IEEE Symposium on Foundations of Computer Science*, 1976, pp. 208–215.

[18] Mikhail Skopenkov, On approximability by embeddings of cycles in the plane. *Topology and its Applications* **134** (2003), 1–22.

[19] Daniel D. Sleator and Paul F. Dietz, Two algorithms for maintaining order in a list, in *Proc. 19th ACM Symposium on Theory of Computing*, 1987, pp. 365–372. Full version in Tech. Rep. CMU-CS-88-113, Carnegie Mellon University, Pittsburgh, PA, 1988.