Recognizing Weakly Simple Polygons*

Hugo A. Akitaya[1], Greg Aloupis[1], Jeff Erickson[2], and Csaba D. Tóth[3]

1 **Department of Computer Science, Tufts University, Medford, MA, USA**
2 **Department of Computer Science, University of Illinois, Urbana-Champaign, IL, USA**
3 **Department of Mathematics, California State University Northridge, Los Angeles, CA, USA**

Abstract

We present an $O(n \log n)$-time algorithm that determines whether a given planar n-gon is weakly simple. This improves upon an $O(n^2 \log n)$-time algorithm by Chang, Erickson, and Xu [4]. Weakly simple polygons are required as input for several geometric algorithms. As such, how to recognize simple or weakly simple polygons is a fundamental question.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases weakly simple polygon, crossing

Digital Object Identifier 10.4230/LIPIcs.SoCG.2016.8

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 by at most 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 (Fig. 1).

Several definitions have been proposed for weakly simple polygons, each formalizing the intuition that a weakly simple polygon does not cross itself. Some of these definitions were unnecessarily restrictive or incorrect; see [4] for a detailed discussion. Ribó Mor [7] proved that a weakly simple polygon with at least three vertices can be perturbed into a simple polygon continuously while preserving the lengths of its edges, and maintaining that no two edges properly cross. Chang et al. [4] gave an equivalent definition for simple polygons in terms of the Fréchet distance (see Section 2), in which a polygon is perturbed into a simple closed curve. The latter definition is particularly useful for recognizing weakly simple polygons. Apart from perturbing vertices, it allows transforming edges into polylines (by subdividing the edges with Steiner points which may be perturbed). The perturbation of a vertex incurs only local changes, and need not affect the neighborhood of adjacent vertices.

It is easy to decide whether an n-gon is simple in $O(n \log n)$ time by a sweepline algorithm [8]. Chazelle's triangulation algorithm recognizes simple polygons in $O(n)$ time, because it only produces a triangulation if the input is simple [5]. Recognizing weakly simple polygons is more subtle. Cortese et al. [6] achieved this in $O(n^3)$-time. Chang et al. [4] improved this to $O(n^2 \log n)$ in general; and to $O(n \log n)$ for several special cases. They

* This work was partially supported by the NSF grants CCF-1408763, CCF-1422311, and CCF-1423615.

32nd International Symposium on Computational Geometry (SoCG 2016).
Editors: Sándor Fekete and Anna Lubiw; Article No. 8; pp. 8:1–8:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

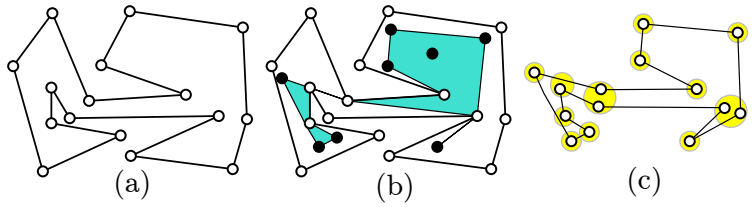

Figure 1 (a) A simple polygon P. (b) Eight points in the interior of P (solid dots); their geodesic hull is a weakly simple polygon P' with 14 vertices. (c) A perturbation of P' into a simple polygon.

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). For polygons with no forks or no spurs, Chang et al. [4] gave an $O(n \log n)$-time algorithm. In the presence of both forks and spurs, their solution is to eliminate forks by subdividing all edges that contain vertices in their interiors, potentially creating a quadratic number of vertices. We show how to manage this situation efficiently, while building on ideas from [4, 6] and from Arkin et al. [2], and obtain the following main result.

▶ **Theorem 1.** *Deciding whether a given n-gon is weakly simple takes $O(n \log n)$ time.*

Our 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 applies known methods 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). We can also perturb any weakly simple polygon into a simple polygon, in $O(n \log n)$ time, by reversing the sequence of operations.

2 Preliminaries

Here, we review definitions from [4] and [6]. We adopt terminology from [4].

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 $\gamma : \mathbb{S}^1 \to \mathbb{R}^2$. 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. 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 $|p_ip'_i| < \varepsilon$ for all $i = 0, \ldots, n-1$. This definition is difficult to work with because a small perturbation of a vertex modifies the neighborhoods of the two adjacent vertices. Chang et al. [4] gave an equivalent definition in terms of the 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 independent of ε, as explained below.

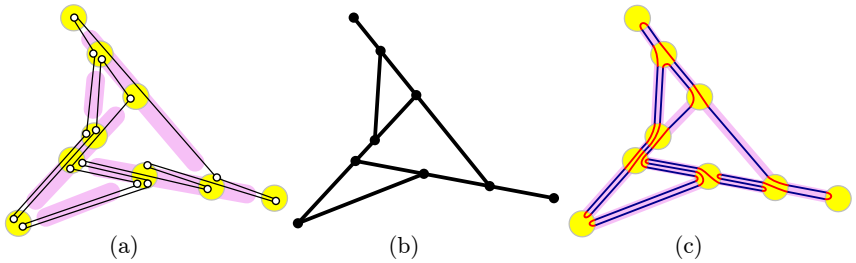

Figure 2 (a) The bar decomposition for a weakly simple polygon P with 16 vertices (P is perturbed into a simple polygon for clarity). (b) Image graph of P. (c) A combinatorial representation of P.

Bar decomposition and image graph. Two edges of a polygon P *cross* if their interiors intersect at precisely one point. The edges of a weakly simple polygon cannot cross. In the following, 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 Fig. 2(a) where equivalence classes are shaded. The union of all edges in an equivalence class is called a *bar*. All bars of a polygon can be computed in $O(n \log n)$ time [4]. The bars are line segments that are pairwise noncrossing and nonoverlapping, and the number of bars is $O(n)$.

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* to distinguish them from the vertices and edges of P. Every node that is not in the interior of a bar is called *sober*. 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 an edge in G if they are consecutive nodes along a bar; see Fig. 2(b). Hence G has $O(n)$ vertices and edges, and it can be computed in $O(n \log n)$ time [4]. 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 modify a polygon and ultimately eliminate forks and spurs. An operation that produces a weakly simple polygon iff it is performed on a weakly simple polygon is called *ws-equivalent*. We shall use some known ws-equivalent operations, and introduce several new ws-equivalent operations in Sections 3.3–5.

Combinatorial characterization of weak simplicity. To show that an operation is ws-equivalent, it suffices to show the existence of ε-perturbations. We will use perfect matchings to combinatorially represent ε-perturbations (independent of ε or any specific embedding). This representation is a variation of the "strip system" introduced in [4].

Let P be a polygon and G its image graph. We construct a family of simple polygons as follows. Let $\varepsilon = \varepsilon(P) \in (0, 1)$, to be specified shortly. For every node u of G, draw a disk D_u of radius ε centered at u. Choose ε sufficiently small so that the disks are pairwise disjoint, and no disk intersects a nonincident segment of G. Let the *corridor* N_{uv} of segment 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, $N_{uv} = \{p \in \mathbb{R}^2 : \mathrm{dist}(p, uv) \le \varepsilon^2, p \notin D_u \cup D_v\}$. Reduce ε further, so that all corridors are pairwise disjoint, and also disjoint from any disk D_w, $w \notin \{u, v\}$. For every segment uv of G, let the *volume* $\mathrm{vol}(uv)$ be the number of edges of P between u and v. For every segment uv, draw $\mathrm{vol}(uv)$ parallel line segments between ∂D_u and ∂D_v within N_{uv}. Finally, for every disk D_u, construct a plane straight-line perfect matching between the segment endpoints

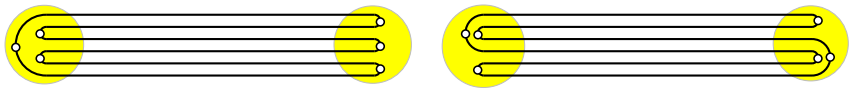

Figure 3 Two perturbations of a weakly simple polygon on 6 vertices (all of them spurs) that alternate between two distinct points in the plane.

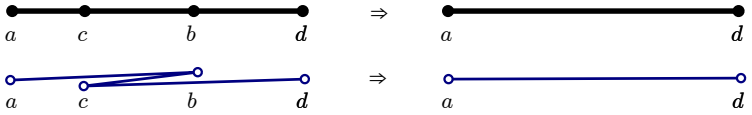

Figure 4 A crimp reduction replaces $[a, b, c, d]$ with $[a, d]$. Top: image graph. Bottom: polygon.

on the boundary ∂D_u (see Fig. 2(c) where the matchings are drawn with circular arcs for clarity). The line segments in the corridors and the perfect matchings in the disks produce a plane 2-regular graph Q. Denote by $\Phi(P)$ the family of all plane graphs constructed in this way such that Q is connected and visits the disks in the same cyclic order as P. By Theorem B.2 in [4], P is weakly simple iff $\Phi(P) \neq \emptyset$. Every $Q \in \Phi(P)$ defines (and is defined by) a linear order on overlapping edges of P.

Note that the above combinatorial representation, which will be used in our proofs, may have $\Theta(n^2)$ size, since each edge passing through a node u contributes one edge to a matching in D_u. We use this simple combinatorial representation in our proofs of correctness, but our algorithm will not maintain it explicitly.

In the absence of spurs, a weakly simple polygon P defines a unique crossing-free perfect matching in each disk D_u [4] which defines a 2-regular graph Q. Consequently, to decide whether P is weakly simple it is enough to check whether $Q \in \Phi(P)$. This is no longer the case in the presence of spurs. In fact, it is not difficult to construct weakly simple n-gons that admit $2^{\Theta(n)}$ combinatorially different perturbations into simple polygons; see Fig. 3.

3 Preprocessing

By a standard line sweep [8], we detect any edge crossing. We then simplify the polygon, using some known steps from [2, 4], and some new. 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 where all edges are collinear. They define the ws-equivalent crimp-reduction operation (see the full paper [1] for details). A *crimp* is a chain of three consecutive 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 strict). The crimp-reduction replaces the crimp with edge $[a, d]$; see Fig. 4.

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 [4]. If we greedily apply crimp-reductions and merge operations (cf. Section 2), 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.

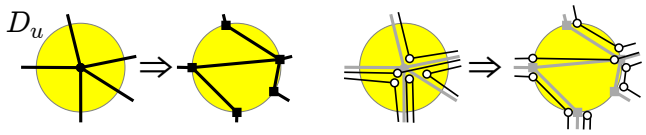

Figure 5 Node expansion. (Left) Changes in the image graph. (Right) Changes in P (the vertices are perturbed for clarity); new nodes are shown as squares.

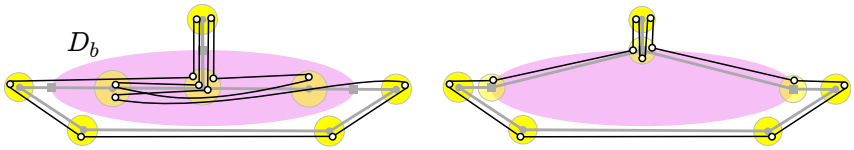

Figure 6 The old-bar-expansion converts a non-weakly simple polygon to a weakly simple one.

▶ **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 any j such that $i < j < k$, consider $|e_j|$. If $|e_{j-1}|$ and $|e_{j+1}|$ are at least as large, 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. ◀

3.2 Node expansion

Compute the bar decomposition of P and its image graph G (defined in Section 2, see Fig. 2). For every sober node of the image graph, we perform the ws-equivalent node-expansion operation, described by Chang et al. [4][Section 3] (Cortese et al. [6] call this a *cluster expansion*). Let u be a sober node of the image graph and D_u be the disk centered at u with radius 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 Fig. 5. If a node expansion produces an edge crossing, report that P is not weakly simple.

3.3 Bar expansion

Chang et al. [4][Section 4] define a bar expansion operation, referred in this paper 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, and replace each maximal path that intersects with D_b by a straight-line edge. If b contains no spurs, old-bar-expansion is known to be ws-equivalent [4]. Otherwise, it can produce false positives, hence it is not ws-equivalent; see Fig. 6 for an example.

New bar expansion operation. Let b be a bar in the image graph with at least one interior node; see Fig. 7. Let D_b be an ellipse whose major axis is in b such that D_b contains all interior nodes of b (all 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, but all interior vertices of b remain at their original positions.

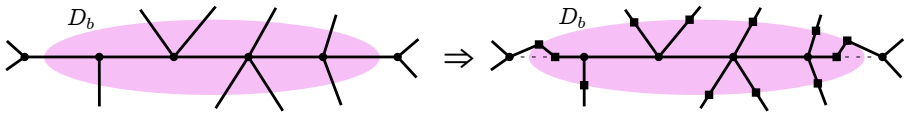

Figure 7 The changes in the image graph caused by new-bar-expansion.

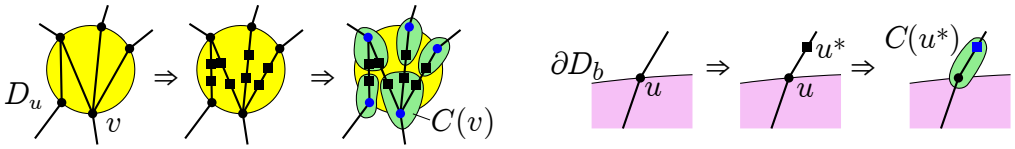

Figure 8 Formation of new clusters around (left) a sober node and (right) a node on the boundary of an elliptical disk. The roots of the induced trees are colored blue.

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.

Terminology. Here, we classify each path 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 $\mathcal{P}$ can be classified based on the position of their endpoints. A path p is called a

- *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.

Let $\mathcal{P}in \subset \mathcal{P}$ be the set of pins, and $\mathcal{V} \subset \mathcal{P}$ the set of V-chains. Let M_{cr} be the set of longest edges of *cross chains* in $\mathcal{P}$ (by Lemma 2, each cross chain contributes one or two edges). Every weakly simple polygon has the following property.

(A3) No edge in M_{cr} lies in the interior of any other edge of P.

We can test property (A3) in $O(n \log n)$ time at preprocessing (for each bar, sort all edges by their endpoints, and compute M_{cr}). If property (A3) fails, we report that P is not weakly simple. The operations introduced in Section 2 maintain properties (A1)–(A3) in bars.

3.4 Clusters

As a preprocessing 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 Fig. 8. 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^*.

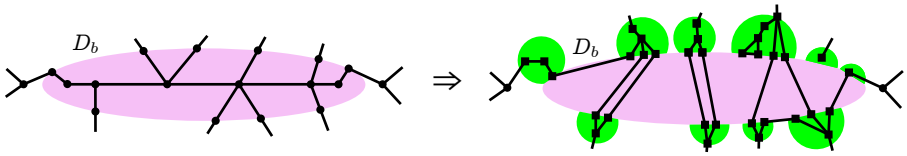

Figure 9 The changes in the image graph caused by a bar simplification.

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.

Additionally, each leaf node ℓ satisfies the following:

(I3) ℓ has degree one or two in the image graph of P;

(I4) there is no spur at ℓ;

(I5) no edge passes through 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) and (I2.b) and every leaf node satisfies (I3)–(I5) since it was created by a subdivision. The operations in Section 4 maintain these invariants.

Dummy vertices. Although the operations described in Sections 4 and 5 introduce nodes in 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 contain $\Theta(n^2)$ dummy vertices in the worst case, however we do not store these explicitly. By (I1), (I2) and (I4) 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 effect of the operations (Section 4.1), define them and show that they are ws-equivalent (Sections 4.2 and 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)–(A3). We summarize the overall effect of the bar simplification subroutine for a given expanded bar.

Changes in the image graph *G*. Refer to Fig. 9. 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.

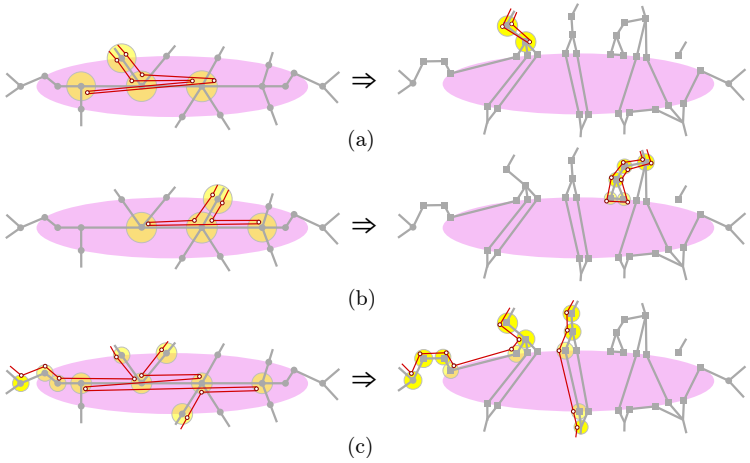

Figure 10 The changes in the polygon caused by a bar simplification.

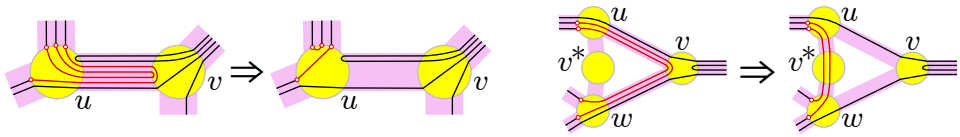

Figure 11 Left: Spur-reduction(u, v). Right: Node-split(u, v, w).

Changes in the polygon *P*. Refer to Fig. 10. Consider a maximal path p in P that lies in D_b. The bar simplification will replace $p = [u, \ldots, v]$ 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 p_1 and $p_2 (= p_1^{-1})$, then p' will be a spur in the cluster containing u (Fig. 10(a)). If p has no such decomposition, but its two endpoints are at the same node, $u = v$, then p' will be a single edge connecting two leaves in the cluster containing u (Fig. 10(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 (Fig. 10(c)).

4.2 Primitives

The operations in Section 4.3 rely on two basic steps, *spur-reductions* and *node splits* (see Fig. 11). The proof of their ws-equivalence is available in the full paper [1]. 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 $[u, v]$. Replace any path $[u, v, u]$, with a single-vertex path $[u]$.
- node-split(u, v, w). Assume segments uv and vw are consecutive in radial order around v, and not collinear with an adjacent segment; and P contains no spurs of the form $[u, v, u]$ or $[w, v, w]$. Create node v^* in the interior of the wedge $\angle(u, v, w)$ sufficiently close to v; and replace every path $[u, v, w]$ with $[u, v^*, w]$.

4.3 Operations

We describe three operations: pin-extraction, V-shortcut, and L-shortcut. The first two eliminate pins and V-chains, respectively, and the third simplifies chains in $\mathcal{P}$ with two or more vertices in the interior of D_b, removing one vertex at a time.

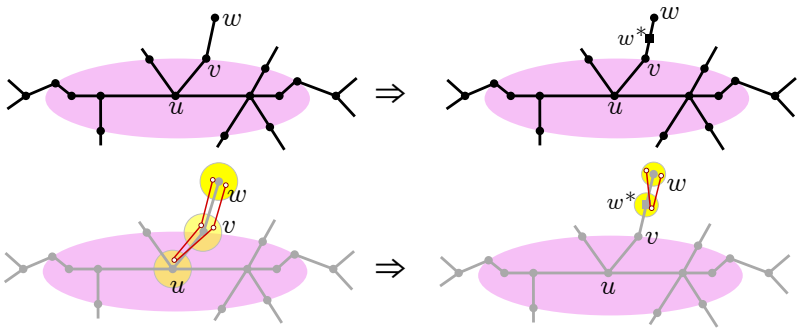

Figure 12 pin-extraction. Changes in the image graph (top), changes in the polygon (bottom).

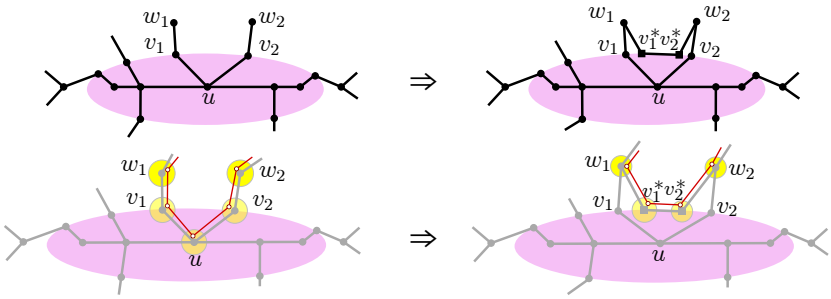

Figure 13 V-shortcut. Changes in the image graph (top), changes in the polygon (bottom).

Pin-extraction and V-shortcut operations. These operations are combinations of primitives and, therefore, they are ws-equivalent. (I1)–(I5) are maintained by construction, and (A1)–(A3) are also maintained within each bar. Proofs are available in the full paper [1].

- pin-extraction(u, v). Assume that P satisfies (I1)–(I5) 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). See Fig. 12 for an example.
- V-shortcut(v_1, u, v_2). Assume that P satisfies (I1)–(I5) 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 u^*; (2) node-split(u^*, v_1, w_1) and node-split(u^*, v_2, w_2); which create $v_1^*, v_2^* \in \partial D_b$; (3) merge every path $[v_1^*, u^*, v_2^*]$ to $[v_1^*, v_2^*]$. See Fig. 13 for an example.

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. 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. Recall that M_{cr} is 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}. We partition L_v into four subsets: 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 $x(u_1) > x(u_2)$;

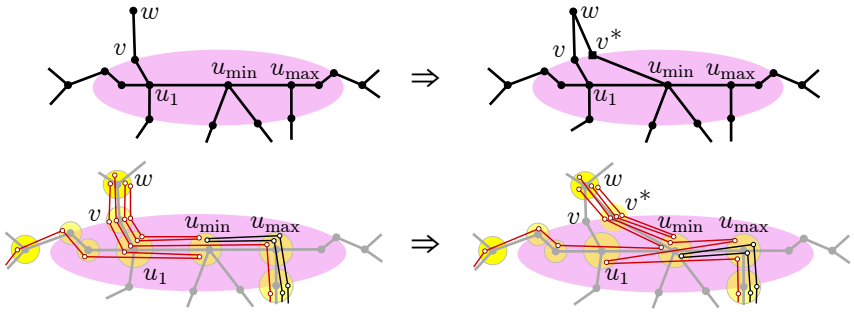

Figure 14 L-shortcut. Changes in the image graph (top), changes in the polygon (bottom).

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)–(I5), $v \in \partial D_b$ 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)** no pins 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 $\angle(v, u_1, u_2)$;
  - **(B3)** no edge $[p, q]$ such that $p \in \partial D_b$ is a top vertex and $q \in b$, $x(u_1) < x(q) < x(u_{\max})$.

  (See the full paper [1] for an justification of these assumptions.) Do the following.
  - **(0)** Create a new node $v^* \in \partial D_b$ to the right of v sufficiently close to v.
  - **(1)** For every path $[v, u_1, u_2] \in L_v^{TR}$ where 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 $u_2 \in U$, by $[w, v^*, u_{\min}, u_2]$.

  See Fig. 14.

▶ **Lemma 3.** *L-shortcut is ws-equivalent and maintains* (I1)–(I5).

Proof Sketch (see [1] for a full proof). W.l.o.g., assume L-shortcut(v, TR) is executed. Phase (1) is ws-equivalent by [2]. The rest of the operation is equivalent to subdividing every path in L_v^{TR} where $u_2 \neq u_{\min}$ into $[v, u_1, u_{\min}, u_2]$, node-split$(v, u_1, u_{\min})$ (which creates u_1^*), node-split(w, v, u_1^*) (which creates v^*) and merging every path $[v^*, u_1^*, u_{\min}]$ to $[v^*, u_{\min}]$. Except for node-split$(v, u_1, u_{\min})$, all primitives satisfy their constraints and therefore are ws-equivalent. It remains to show that (B1)–(B3) ensure that this primitive is ws-equivalent. Let P' be obtained from P after node-split$(v, u_1, u_{\min})$. If P' is weakly simple, by changing its embedding we can move u_1^* arbitrarily close to u_1 without affecting weak simplicity, hence P is weakly simple. It remains to show that if P is weakly simple, there exists $Q \in \Phi(P)$ such that the paths in L_v^{TR} are the topmost paths in the linear order induced by Q. Indeed, if there is one edge $[p, q]$ above one path in L_v^{TR}, by (B1)–(B3), it must be part of a path $[p, q, r]$ such that q is a spur and $x(u_{\max}) < x(p)$ and $x(u_{\max}) < x(r)$. Then, it can always be moved below the lowest edge $[u_2, u_3]$ adjacent to a path in L_v^{TR} without introducing any crossing (similar to crimp reduction; see Fig. 15). Phase (1) ensures that this is always possible, since after that phase every path in L_v^{TR} has an adjacent edge

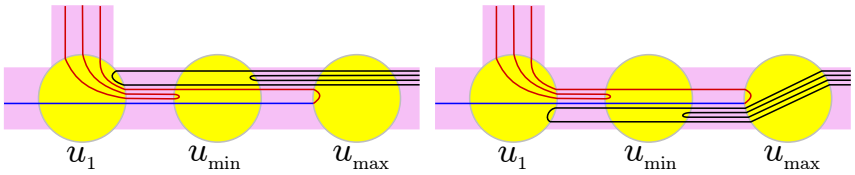

Figure 15 If P_1 is weakly simple, we can change the linear order of the edges as shown.

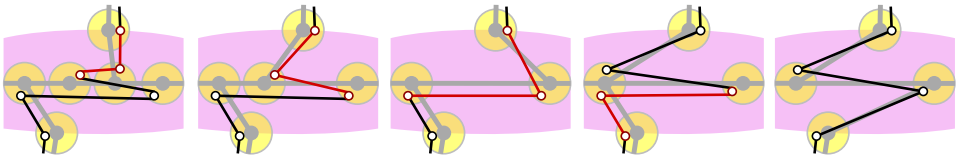

Figure 16 Life cycle of a cross chain in the while loop of bar-simplification. The steps applied, from left to right, are: (4), (3), (4), (6).

on b. Notice that phases (2) and (3) restore (A2) in the bar. Therefore we can "shorten" the lengths of paths in L_v^{TR} to create a simple polygon $Q' \in \Phi(P')$, hence P' is weakly simple. ◀

4.4 Bar simplification algorithm

In this section, we show that the three operations (pin-extraction, V-shortcut, and L-shortcut) can successively remove all spurs of the polygon P from a bar b.

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, and perform V-shortcut(v_1, u, v_2).
- **(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 $L_{v'}^{TL} = \emptyset$ for all $v' \in \partial D_b$, $x(v) < x(v')$ and $x(u_1') < x(u_{\max})$, where u_1' is the unique neighbor of v' on b, do L-shortcut(v, TR).
  - **(c)** Else let v' be the leftmost node such that $x(v) < x(v')$ and $L_{v'}^{TL} \neq \emptyset$. If $L_{v'}^{TL}$ satisfies (B3) do L-shortcut(v', TL), otherwise halt and report that P is not weakly simple.
- **(iv)** Else if there exists $v \in \partial D_b$ such that $L_v^{TL} \neq \emptyset$, repeat steps (iiia-c) with left–right and TR–TL interchanged. (Note the use of L_v instead of $\widehat{L_v}$. The same applies to (vi)).
- **(v)** Else if there exists $v \in \partial D_b$ such that $\widehat{L_v}^{BL} \neq \emptyset$, repeat steps (iiia-c) using BL and BR in place of TR and TL, respectively, and left-right interchanged.
- **(vi)** Else if there exist $v \in \partial D_b$ such that $L_v^{BR} \neq \emptyset$, repeat steps (iiia-c) using BR and BL in place of TR and TL respectively.

After the loop ends, perform old-bar-expansion (cf. Section 3.3) in the ellipse D_b;
Return P (end of algorithm).

Informally, bar-simplification "unwinds" each polygonal chain in the bar, while extracting pins and V-chains as they appear, by alternating between steps (3) to (6) (see Fig. 16). Step (3) uses $\widehat{L_v}^{TR}$ (instead of ${L_v}^{TR}$) to avoid an infinite loop.

▶ **Lemma 4.** *The operations performed by* bar-simplification(P, b) *are ws-equivalent, and maintain properties* (A1)–(A3) *and* (I1)–(I5) *inside* D_b*. 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 (1)–(2). Since every pin can be extracted from a polygon satisfying (I1)–(I5), 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 condition (B1) 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). This edge is part of a path $[p, u, q]$. Node q must be on ∂D_b between v_1 and v_2, otherwise paths $[p, u, q]$ and $[v_1, u, v_2]$ cross. However, $p \neq q$, otherwise $[p, u, q]$ would be a pin. Consequently, $[p, u, q]$ is a V-chain where $|x(p) - x(q)| < |x(v_1) - x(v_2)|$, contrary to the choice of $[v_1, u, v_2] \in \mathcal{V}$. This confirms that V-shortcut(v_1, u, v_2) satisfies (B1). Henceforth, assume that $\mathcal{P}in = \emptyset$ and $\mathcal{V} = \emptyset$.

Step (3)–(4). By symmetry, we consider only step (3). We distinguish between two cases.

Case 1: the conditions of (2) are satisfied. We need to show that L-shortcut(v, TR) satisfies (B1)–(B3). Since $\mathcal{P}in = \emptyset$, condition (B1) is met. Suppose there is an edge $[p, u_1]$ such that segment pu_1 is in the interior of the wedge $\angle(v, u_1, u_{\min})$. Clearly, $p \in \partial D_b$ is a top node. Then edge $[p, u_1]$ is part of a path $[p, u_1, q]$. However, q must be in the closed wedge $\angle(v, u_1, u_{\min})$ otherwise there would be a node-crossing at u_1. Also, q cannot be a top vertex on ∂D_b since $\mathcal{P}in = \mathcal{V} = \emptyset$, and q cannot be on b by the choice of node v. This confirms (B2). We argue similarly for (B3). Suppose there is an edge $[p, q]$ such that $p \in \partial D_b$ is a top vertex and $q \in b$, $x(u_1) < x(q) < x(u_{\max})$. This edge is part of a path $[p, q, r]$. Node r must be on or above b, otherwise there would be a node-crossing at q. It cannot be a top vertex, since $\mathcal{P}in = \mathcal{V} = \emptyset$. It cannot be to the left of q, otherwise the conditions of (2) are satisfied; and it cannot be to the right of q, otherwise $L_p^{TR} \neq \emptyset$ with $x(v) < x(p)$, contrary to the choice of v. This confirms that L-shortcut(v, TR) satisfies (B2)–(B3) and can be performed.

Case 2: the conditions of (2) are not satisfied. Let the path $[v', u'_1, u'_{\min}] \in L_{v'}^{TL}$ be selected in L-shortcut(v', TL) by the algorithm. Condition (B2) is satisfied similar to Case 1. If (B3) fails, there is an edge $[p, q]$ such that $p \in \partial D_b$ is a top vertex and $q \in b$, $x(u'_{\max}) < x(q) < x(u_{\max})$ (Recall that left and right are interchanged in L^{TL}). Edge $[p, q]$ is part of a path $[p, q, r]$, where $r \in b$, similar to Case 1. This implies $[p, q, r] \in L_p^{TR} \cup L_p^{TL}$. If $x(v) < x(p) < x(v')$, then either $L_p^{TR} \neq \emptyset$, which contradicts the choice of v, or $L_p^{TL} \neq \emptyset$, which contradicts the choice of v'. Consequently, $x(p) \leq x(v)$. This implies $x(u'_{\max}) < x(p) \leq x(v)$, so the paths $[v, u_1, u_{\max}]$ and $[v', u'_1, u'_{\max}]$ cross. Therefore the algorithm correctly finds that P is not weakly simple.

Steps (5)–(6). If steps (1)–(4) do not apply, then $\widehat{L_v}^{TR} \cup L_v^{TL} = \emptyset$. 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 (5)–(6) do not change these properties. Consequently, once steps (5)–(6) are executed for the first time, steps (3)–(4) are never executed again. By a symmetric argument steps (5)–(6) eliminate all paths in $\widehat{L_v}^{BL} \cup L_v^{BR}$. If the while loop terminates, every edge in

b is also in M_{cr} and $L_v^{TL} \cup L_v^{BR} = \emptyset$. Consequently, by Lemma 2, b contains no spurs and old-bar-expansion is ws-equivalent. This 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 (3)–(4), and at most once in steps (5)–(6). W.l.o.g., consider step (3). We apply an L-shortcut in two possible cases. We show that it does not introduce crimps in Case 2. In step (3), we only perform L-shortcut(v', TL) if (B3) is satisfied and $x(u_1') < x(u_{\max})$. So for all $[v', u_1', u_2'] \in L_{v'}^{TL}$, we have $x(u_1) < x(u_2')$. Suppose, for contradiction, that $[u_1', u_2']$ is the only longest edge of some cross chain (and hence L-shortcut would introduce a crimp). Then, $[u_1', u_2'] \in M_{cr}$ is inside $[u_1, u_{\max}]$, contradicting (A3).

Consider Case 1. Notice that L-shortcut(v, TR) is executed only if there exists a top node p with $x(p) < x(u_1)$ such that $\widehat{L_p}^{TR} \neq \emptyset$. Suppose that L-shortcut(v, TR) introduces a crimp in the path $[v, u_1, u_2] \in L_v^{TR}$. This operation removes this subpath of a cross chain from L_v^{TR}, but introduces $[v^*, u_2, u_1]$ into $L_{v^*}^{TL}$. By the time the algorithm executes L-shortcut(v^*, TL), we know that for every top vertex p with $x(p) < x(u_1)$, $\widehat{L_p}^{TR} = L_p^{TL} = \emptyset$. This implies that, after L-shortcut(v^*, TL) is performed, although a path $[v^{**}, u_1, u_2]$ is introduced in $L_{v^{**}}^{TR}$, L-shortcut(v^{**}, TR) can never be performed. The same arguments apply to steps (5)–(6). ◀

▶ **Lemma 5.** *Algorithm* **bar-simplification**(P, b) *takes* $O(m \log m)$ *time, where* m *is the number of vertices in* b.

Proof. pin-extraction, V-shortcut, and L-shortcut each make $O(1)$ changes in the image graph. pin-extraction and V-shortcut decrease the number of vertices inside D_b. Each L-shortcut does as well, but they may jointly create $2|\mathcal{P}| = O(m)$ crimps, by Lemma 3. So the total number of operations is $O(m)$. When $[v, u_1, u_2] \in L_v^{TR}$ and $u_2 \neq u_{\min}$, L-shortcut replaces $[v, u_1, u_2]$ by $[v^*, u_{\min}, u_2]$: $[u_1]$ 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 $\Theta(m^2)$. 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_v^X, 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_v^X, the first two nodes in the paths $[v, u_1, u_2] \in L_v^X$ are the same by (I3), 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 $[v, u_1, u_2] \in L_v^X$ whether it is part of a cross chain, and whether $[u_1, u_2]$ is the only longest edge of that chain.

Steps (1)-(2) 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 one of steps (3)–(4) which can be generalized to other occurrences of the L-shortcut operation. The elements $[v, u_1, u_{\min}] \in L_v^{TR}$ are simplified to $[v^*, u_{\min}]$. For each of these paths, say that the next edge along P is $[u_{\min}, u_3]$. Then, the paths $[v^*, u_{\min}, u_3]$ are inserted into either $\mathcal{P}in \cup \mathcal{V}$ if $u_3 \in \partial D_b$ is a top vertex, or $L_{v^*}^{TL}$ if $u_3 \in b$. We can find each chain $[v, u_1, u_{\min}] \in L_v^{TR}$ in $O(1)$ time since L_v^{TR} is sorted by $x(u_2)$. Finally, all other paths $[v, u_1, u_2] \in L_v^{TR}$, where $u_2 \neq u_{\min}$, become $[v^*, u_{\min}, u_2]$ and they form the new set $L_{v^*}^{TR}$. Since we store only the last vertex $[u_2]$, which is unchanged, we create $L_{v^*}^{TR}$ 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

When there are no forks in the polygon, we can decide weak simplicity using [4][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. Here, we show how to eliminate spurs in $O(n \log n)$ time. After the bar simplification phase, each vertex of P belongs to a cluster.

Formation of Groups. Recall that by (I1) each cluster induces a tree. We first modify the image graph, transforming each tree into a binary tree by adding children to nodes with degree higher than 3. This does not affect the benchmark representation and is ws-equivalent (it can be reversed by node-splits and merges). By construction, if a segment uv connects nodes in different clusters, both u and v are leaves or both are root nodes. We define a *group* G_{uv} as the set of two clusters $C(u)$ and $C(v)$ if their roots are connected by the segment uv.

Recall that we only store benchmark vertices in each cluster. We denote by $[u_1; \ldots; u_k]$ (using semicolons) a path inside a group defined by the benchmark vertices $u_1, \ldots, u_k$. Let $\mathcal{B}$ be the set of paths between two consecutive benchmark vertices in G_{uv}. By invariants (I1), (I2) and (I4), every path in $\mathcal{B}$ has one endpoint in $T[u]$ and one in $T[v]$ and every spur in G_{uv} is incident to two paths in $\mathcal{B}$.

Overview. Assume that $\mathcal{G}$ is a partition of the nodes of the image graph into groups satisfying (I1)–(I5). 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 the total number of benchmark vertices.

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 $T[u]$.
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. The node w is the new root of the tree.
   b. If u contains spurs, eliminate them as described below.
   c. If u contains no spurs, split G_{uv} into two groups along a chain of segments starting with uv as described below. Rename a largest resulting group to G_{uv}.

The detailed description of steps 2b and 2c, as well as the analysis of the algorithm and the supporting data structures are in the full paper [1]. Here we give a brief summary. Step 2b first replaces every path of the form $[t_1; u; t_2]$ by a path $[t_1; t_2]$ (Fig. 17(a)–(b)). The resulting new path $[t_1; t_2]$ passes through the lowest common ancestor of t_1 and t_2, denoted $\mathrm{lca}(t_1, t_2)$. Notice that t_1 and t_2 belong to $T[v]$, therefore $[t_1; t_2]$ does not satisfy (I2). We complete Step 2b by a sequence of "repair" operations that restore (I2). One is analogous to pin-extraction, moving a spur from a leaf of $T[v]$ into an adjacent group (Fig. 17(b)–(c)). The other is analogous to V-shortcut: it creates a new group for each node ℓ where $\ell = \mathrm{lca}(t_1, t_2)$ for some path $[t_1; t_2]$ that violates (I2). The set of all $[t_1; t_2]$ for which $\mathrm{lca}(t_1, t_2) = \ell$ induces

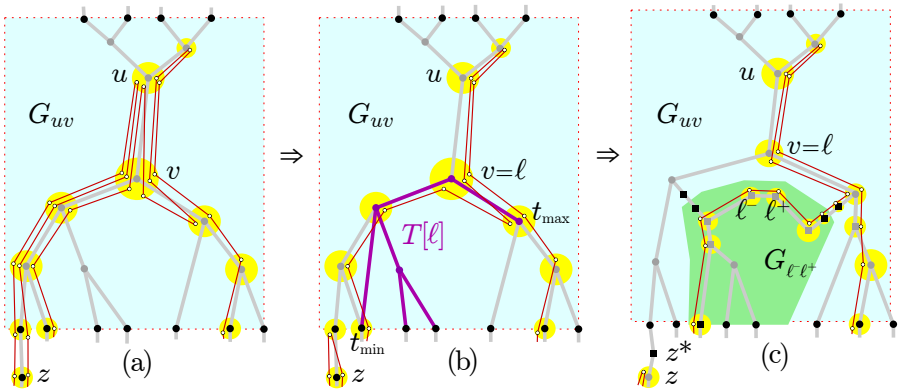

Figure 17 (a) u contains spurs. (b) After eliminating spurs, $T[v]$ does not satisfy (I2). (c) The analogues of pin-extraction and V-shortcut. Leaf nodes are shown black.

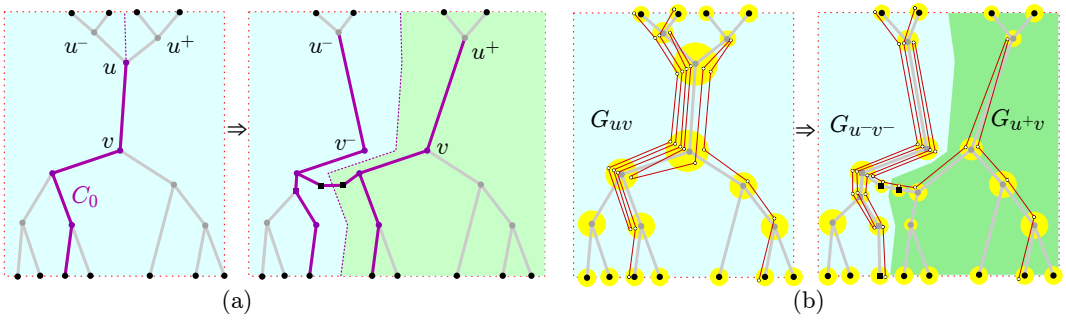

Figure 18 Splitting group G_{uv}. (a) Changes in the image graph. (b) Changes in the polygon.

a tree $T[\ell]$, in which t_1 and t_2 are in the left and right subtree of ℓ, respectively. We then remove all such paths from $T[v]$ and create a new group that satisfies (I1)–(I5).

Step 2c splits G_{uv} into two groups along a chain of segments C_0 (Fig. 18(a)–(b)). The tree $T[u]$ naturally splits into the left and right subtrees, $T[u^-]$ and $T[u^+]$, but splitting $T[v]$ is a nontrivial task. $\mathcal{B}$ is partitioned into $\mathcal{B}^-$ and $\mathcal{B}^+$, paths with one endpoint in $T[u^-]$ and $T[u^+]$, respectively. C_0 represents the shared boundary between $\mathcal{B}^-$ and $\mathcal{B}^+$ and can be found in $O(\log|\mathcal{B}|)$ time. W.l.o.g., $|\mathcal{B}^-| < |\mathcal{B}^+|$. We build a tree $T[v^-]$ induced by $\mathcal{B}^-$ and adjust $T[v]$ so that it is induced by $\mathcal{B}^+$. This is accomplished in $O(|\mathcal{B}^-|)$ time by deleting from $T[v]$ nodes unique to $T[v^-]$. The spurs connected to one path in $\mathcal{B}^-$ and one in $\mathcal{B}^+$ are replaced by a pair of benchmarks on the boundary of the two groups. By a heavy path decomposition argument, the overall time spent in this step is $O(n \log n)$.

6 Conclusion

We presented an $O(n \log n)$-time algorithm for deciding whether a polygon with n vertices is weakly simple. The problem has a natural generalization for planar graphs [4]. It is an open problem to decide efficiently whether a drawing of a graph H is weakly simple, i.e., whether a drawing P of H is within ε Fréchet distance from an embedding Q of H, for all $\varepsilon > 0$.

We can also generalize the problem to higher dimensions. A polyhedron can be described as a map $\gamma : \mathbb{S}^2 \to \mathbb{R}^3$. 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 [3] 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 ($g^{-1}(x)$, for a point $x \in P$, maps to two points in different 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.

References

1 Hugo A. Akitaya, Greg Aloupis, Jeff Erickson, and Csaba D. Tóth. Recognizing weakly simple polygons. Preprint, 2016. `arXiv:1603.07401`.

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*, 29(1):23–46, 2004.

3 Marshall Bern and Barry Hayes. The complexity of flat origami. In *Proc. 7th ACM-SIAM Sympos. on Discrete Algorithms*, pages 175–183, 1996.

4 Hsien-Chih Chang, Jeff Erickson, and Chao Xu. Detecting weakly simple polygons. In *Proc. 26th ACM-SIAM Sympos. on Discrete Algorithms*, pages 1655–1670, 2015.

5 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(3):485–524, 1991.

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

7 Ares Ribó Mor. *Realization and counting problems for planar structures*. PhD thesis, Freie Universität Berlin, Department of Mathematics and Computer Science, 2006.

8 Michael Ian Shamos and Dan Hoey. Geometric intersection problems. In *Proc. 17th IEEE Sympos. Foundations of Computer Science*, pages 208–215, 1976.