Recognizing Weak Embeddings

A dissertation

submitted by

Hugo Alves Akitaya,

B.Eng., Universidade de Brasília (2011)

M.Eng., University of Tsukuba (2014)

In partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Science

TUFTS UNIVERSITY

August 2018

Thesis Supervisors: Prof. Diane L. Souvaine, Prof. Csaba D. Tóth, and

Prof. Erik D. Demaine

To my parents who always believed in me.

Acknowledgments

Funding for my research was provided by NSF grants CCF-1422311 and CCF-1423615, and the Science Without Borders scholarship program. My thanks to the very kind staff at LASPAU who administered my Science Without Borders scholarship. I'd also like to thank Janelle and Kirk Loevner for their generous support through the Loevner Fellowship for the 2016-2017 academic year.

I thank all my advisors for the academic guidance, support, and friendship along the way. This includes, of course, my primary advisor Diane, who encouraged and trusted in me, Csaba and Erik, for guiding me academically, and also Greg and Marty who I consider to be my extra-official advisors. Many thanks to all of my coauthors and collaborators. This research couldn't have happened without you. My special thanks to the participants of the MIT-Tufts Computational Geometry Group notably Zach Abel, Pesto Hesterberg, Jayson Lynch and Jason Ku.

Last but not least, I thank my friends and extended family, including Brian, J.M., David, and my friends in Japan, for keeping me sane and making me laugh at impossible times. A special mention for my brother and oldest friend, Lucas (my brother from another mother), and Anika (the tiny foxtato that believed that I could do the thing).

Hugo Alves Akitaya

TUFTS UNIVERSITY

August 2018

Recognizing Weak Embeddings

Hugo Alves Akitaya

Thesis Supervisors: Prof. Diane L. Souvaine, Prof. Csaba D. Tóth, and Prof. Erik D. Demaine

An *embedding* of a graph G is a drawing $\varphi : G \to M$ of G on a surface M such that every vertex in $V(G)$ is mapped to a distinct point and edges in $E(G)$ to interior-disjoint Jordan arcs between the corresponding vertices. A *weak embedding* is a map $\varphi : G \to M$ such that, for every $\varepsilon > 0$, there exists an embedding $\psi_\varepsilon : G \to M$ so that $\|\varphi - \psi_\varepsilon\| < \varepsilon$, where $\|.\|$ is the uniform norm (i.e., sup norm). We say that the embedding ψ_ε *approximates* φ. The study of weak embeddings lies at the interface of several independent lines of research in mathematics and computer science. In particular, it generalizes various graph visualization models and is a special case of the notoriously difficult cluster-planarity problem. It is also related to the foldability problem in computational origami. This thesis presents several results on the algorithmic complexity of recognizing weak embeddings.

Chapter 2 describes an $O(n \log n)$-time algorithm that recognizes weak embeddings in $\mathbb{R}^2$ when G is regular degree-2. This generalizes the problem of recognizing weakly simple polygons (whether a polygon is approximable by a simple polygon) and improves the previous upper bound of $O(n^2 \log n)$.

Chapter 3 is about the recognizing weak embeddings of general graphs. The first algorithm that solves this problem (when G is an arbitrary graph) was given by Fulek and Kynčl (2017) and runs in $O(m^{2\omega}) \leq O(m^{4.75})$, where $\omega \in [2, 2.373)$ is the matrix multiplication exponent. Chapter 3 describes an $O(n^2 \log n)$-time algorithm

for the problem based on local operations. The algorithm runs in $O(n \log n)$ time if the input is a simplicial map.

A simplicial complex is a generalization of a graph. A graph can be seen as an abstract simplicial 1-complex, i.e., a set of points (0-dimensional simplices) connected by line segments (1-dimensional simplices). A simplicial 2-complex can be defined by a simplicial 1-complex and a set of triangles (2-dimensional simplices) each attached to a cycle of three edges. Chapter 4 shows that it is NP-complete to decide whether a map $\varphi : A \to \mathbb{R}^3$ is a weak embedding of a simplicial 2-complex. We first reduce NAE-3SAT to a computational origami problem called Flat-Foldability, also strengthening the previously known results for this problem. Then we reduce from Flat-Foldability to recognition of weak embeddings.

Contents

List of Figures

Chapter 1

Introduction

This chapter provides an overview of this thesis and describes each subsequent chapter. In order to do so, the chapter also provides some informal definitions. Each technical chapter provides formal definitions as needed.

1.1 Intuitive Definition

Folding is present in our daily lives in many ways. Just to name a few examples, packaging, the relative position of the keyboard and the screen of your laptop, and even where your arm is relative to your body can be expressed in terms of folding. Usually, two objects can't occupy the same space at the same time, but in the mathematics of folding it is convenient to model 3-dimensional objects with lower-dimensional approximations which allow objects to overlap. For instance, although thickness can play an important role in paper folding, we usually think of paper as a 2-dimensional zero-thickness material. Imagine a square piece of paper folded in half along its diagonal, see Figure 1.1. We describe this folded state as being flat, i.e., the two triangles that make up the square paper lie on the same plane and occupy the same space. Despite this overlap, we know that this representation approximates a 3-dimensional state where the paper does not go through itself. We call this approximation of a non-crossing state a *weak embedding*.

In order to more accurately model what really happens with the paper we

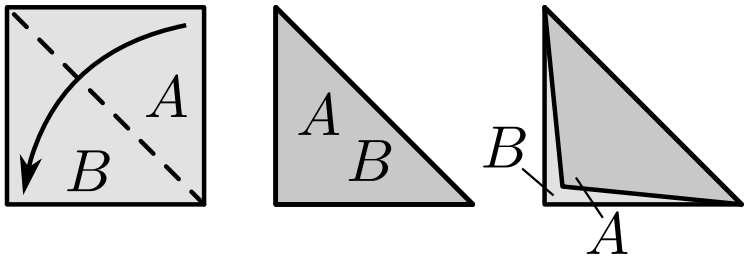


Figure 1.1: Folding a square paper along one of its diagonals. (Middle) A weak embedding where triangles A and B overlap and (right) its perturbation in which the paper does not intersect itself.

might also want the information that triangle A is on top of triangle B. With this information, we can perturb our flat approximation of the folded state into the same position as the real paper by moving each point of our mathematically ideal paper by some small amount and making sure that the paper will not cross itself in this new state.

This thesis talks about the problem of recognizing weak embeddings. Given a weak embedding of an object (which might have parts of the object occupying the same space), we want to obtain a non-crossing state where each point occupies a unique position by moving each point by a small amount. We also want to report the case when we are given a state that does not approximate any real folded state of the object, i.e., every small perturbation of this input has the object crossing itself.

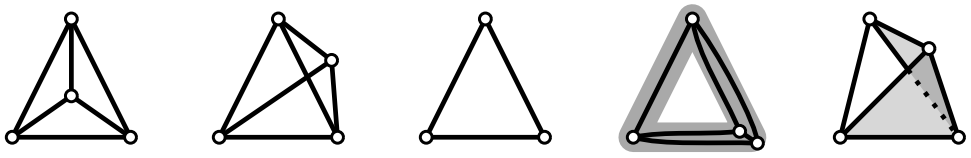
1.2 Simplicial Complexes and (Weak) Embeddings

This thesis presents results on weak embeddings of simplicial complexes, more specifically of 1 and 2 dimensions. In geometry, a simplex is a generalization of a triangle to k dimensions. We call a simplex in k dimensions a *k-simplex*. We will focus only on 0-simplices, 1-simplices, and 2-simplices, which are respectively *points*, *line segments*, and *triangles*. A simplicial complex is a set of simplices that intersect only at simplices. A simplicial k-complex is a simplicial complex that contains simplices of dimensions k and lower. We start by defining an *abstract* simplicial complex which

is purely combinatorial, and then define its *geometric realization* which is a drawing of the simplicial complex without crossings and overlaps. An abstract simplicial 1-complex is equivalent to a simple undirected graph. A graph G is defined by a set of vertices, denoted by $V(G)$, and a set of edges, denoted by $E(G)$, which are pairs of distinct vertices in $V(G)$. An abstract simplicial 2-complex (G, T) is equivalent to a graph G and a set T of unordered triples of vertices where each element of T is defined by a cycle of length 3 in G.

A geometric realization of a simplicial 1-complex associates vertices in $V(G)$ with distinct points and edges in $E(G)$ with interior disjoint line segments. The endpoints of the line segment associated with edge uv must be the points associated with vertices u and v. Figure 1.2(a) shows a geometric realization of the complete graph K_4 with four vertices, where the vertices in $V(K_4)$ are shown as four small circles and the edges in $E(K_4)$ are shown as line segments between every pair of vertices. A geometric realization of a simplicial 2-complex (G, T) is a geometric realization of G with interior-disjoint triangles for each element of T. Figure 1.2(e) shows a 2D representation of a 3D drawing of a simplicial 2-complex (K_4, T), where T contains two triangles shown in transparent gray.

The union of points in a geometric realization of a simplicial complex defines a topological space, i.e., a set of points and neighborhoods for each point. Intuitively, this captures, in the most general way, how the complex is connected while associating vertices to points, edges to 1D curves and triangles to 2D surfaces. Since



(a) (b) (c) (d) (e)

Figure 1.2: (a) An embedding of $G = K_4$ in $\mathbb{R}^2$. (b) A drawing of G in $\mathbb{R}^2$ that is neither an embedding nor a weak embedding. (c) A weak embedding $\varphi : G \to \mathbb{R}^2$. (d) An embedding ψ_ε obtained by moving every point in $\varphi(G)$ by at most ε. The ε-neighborhood of $\varphi(G)$ is shown in gray. (e) An embedding of a simplicial 2-complex in $\mathbb{R}^3$.

all topological spaces obtained from a simplicial complex are equivalent (independent of the geometric realization), we use *simplicial complex* to refer to the abstract simplicial complex and its topological space interchangeably.

An *embedding* is a generalization of a geometric realization of a simplicial complex. Figures 1.2(a) and (e) are also examples of embeddings. The difference is that an embedding allows each edge to be realized as a chain of line segments. For a simplicial complex A, an *embedding* is a continuous map $\psi : A \to M$ that is both piecewise linear (i.e., every edge is mapped to a chain of line segments) and injective (i.e., every point in A is mapped to a unique point in M). Here, M is a manifold equipped with a metric (i.e., a space that resembles the Euclidean space near each point) of higher or equal dimension than A. We call a 2-dimensional manifold a *surface*. The genus of a surface is the number of "holes" it has. For example, a sphere is a surface of genus 0 while a torus is a surface of genus 1. An embedding of a graph in a fixed 2-dimensional manifold can be found in linear time [Moh99]; however, if the surface is not fixed, it is NP-hard to find the smallest genus of a surface in which an input graph can be embedded [Tho89]. It was recently proved that finding an embedding of a simplicial 2-complex in $\mathbb{R}^3$ is NP-hard [dMRST18].

A *weak embedding* $\varphi : A \to M$ can be seen as a *folding* of the simplicial complex A in M, where A can "touch" but not "cross" itself. More formally, φ is a weak embedding if it is a continuous piecewise-linear map, and, for every $\varepsilon > 0$, there is an embedding $\psi_\varepsilon : A \to M$ with $\|\varphi - \psi_\varepsilon\| < \varepsilon$, where $\|.\|$ is the uniform norm (i.e., sup norm). Informally, for every $\varepsilon > 0$, we can "perturb" a weak embedding φ and obtain an embedding ψ_ε by moving each point by at most ε (see Figure 1.2(c–d)). In some cases, it is easy to tell whether φ is a weak embedding: Every embedding is a weak embedding; and if the images of two different edges (or triangles) cross transversely, then φ is not a weak embedding (see Figure 1.2(b)). The problem becomes challenging when φ is not injective, i.e., when several edges (and triangles in the 2-dimensional case) overlap.

1.3 Related Work

Origami. The recognition of weak embeddings relates to *computational origami*, which is the study of origami-related problems in computer science. Origami is the art of folding paper, a 2-dimensional surface, to create a (usually 3-dimensional) model representing some artistic subject. Although cuts in the paper are used in traditional models, modern origami artists are usually discouraged from cutting the paper [Hat11]. Even with this stronger restriction, the use of new design techniques, many of which use mathematics, brought the art to a new level, with intricate models that allow an amazing degree of detail [Lan03]. The mathematical and computational studies of origami have also found diverse uses beyond the arts. To name a few, origami-related mathematical properties and algorithms can be used in reconfigurable and self-folding robotics [HAB+10], deployable structures [EHB04, SKSG13], biomedical devices [KTY+06, LPL+13, JKX+13], and airbag simulation [Cro07]. The connections between origami and *mechanical linkages* (an assembly of rigid bodies connected to produce a specific movement) has also been studied [DO07]. Informally, in most mathematical models of origami, a *folded isometry* f is a map from a paper P of d dimensions to $\mathbb{R}^{d+1}$ that does not stretch or tear the paper. A *folded state* is a folded isometry together with an ordering between every pair of overlapping points of P that guarantees that the paper does not "self-cross". The definition of self-crossing is equivalent to the existence of an injective "ε-perturbation", i.e., whether we can obtain a continuous map arbitrarily close to f where each point of the paper maps to a unique point in $\mathbb{R}^{d+1}$ (separating the overlapping layer of paper, which defines the ordering). The Foldability problem asks whether there exists a folded state for a given folded isometry. The problem of recognizing weak embeddings generalizes Foldability. Chapters 2 and 4 will review previous results on the Foldability problem that are used in this thesis.

Topology and Graph Drawing. The study of weak embeddings also lies at the interface of several independent lines of research in mathematics and computer science. In topology, the study of weak embeddings were initiated by Sieklucki [Sie69] in

the 1960s. Weak embeddings of graphs also generalize various graph visualization models such as the recently introduced *strip planarity* [ADLDBF17] and *level planarity* [JLM98]; and can be seen as a special case [AL16] of the notoriously difficult *cluster-planarity* (for short, *c-planarity*) problem [FCE95a, FCE95b], whose tractability remains elusive despite many attempts by leading researchers. Graph visualization models and drawing algorithms have numerous applications in many areas such as data visualization, circuit design, virtual reality, biology, and chemistry [HMM00]. Moreover, weak embeddings are relevant for applications in clustering, cartography, and visualization, where nearby vertices and edges are often bundled to a common node or arc, due to data compression or low resolution.

1.4 Problem Definition

This section defines the problem of recognizing whether a drawing φ of a graph G is a weak embedding. Section 1.5 uses this definition to describe the results presented in this thesis. More in-depth definitions will be given in each chapter.

Let $\varphi : G \to M$ be a continuous piecewise-linear map of a graph G in a 2-dimensional manifold M. Recognizing whether φ is a weak embedding turns out to be a purely combinatorial problem independent of the global topology of the manifold M and the neighborhood of $\varphi(G)$ (as noted in [CEX15]). The key observation here is that we are looking for an embeddding in a small neighborhood of the image $\varphi(G)$, which can be considered as the embedding of some graph H with vertex set $V(H) = \{\varphi(v) \mid v \in V(G)\}$, i.e., the set of points in M containing the image of all vertices in $V(G)$. We can consider H as the *image graph* of φ, an embedded graph in M that contain all points in $\varphi(G)$ and no other point in M. To avoid confusion, we call vertices and edges of H *clusters* and *pipes* respectively.

As such, we can replace M with a neighborhood of H in the formulation of the problem. The *strip system* $\mathcal{H}$ of H (or *thickening* of H) is a 2-manifold obtained by placing pairwise disjoint disks D_u on every cluster $u \in V(H)$ and connecting the disks D_u and D_v with pairwise disjoint rectangles R_{uv} for every pipe $uv \in E(H)$ (a

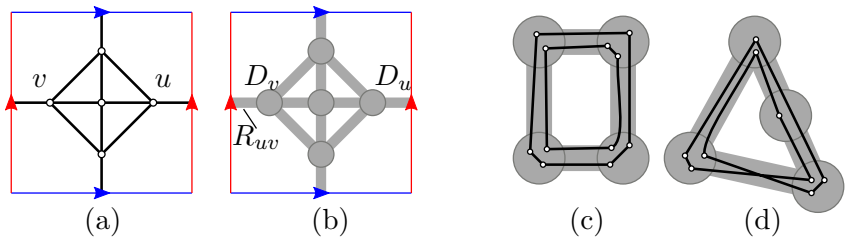


Figure 1.3: (a) An embedding of $H = K_5$ in the torus. (b) Strip system $\mathcal{H}$ of the embedding of H. (c) A weak embedding where G is disconnected, $H = C_4$, and φ is a simplicial map. (d) A negative instance where $G = C_9$, $H = C_3$, and φ is not a simplicial map.

precise definition is given in Chapter 2). See Fig. 1.3(a)–(b).

We formulate an instance of the problem of recognizing weak embeddings as a function $\varphi : G \to H$, where G is an abstract graph, and H is an embedded graph. The map φ is a *weak embedding* if we can embed G in $\mathcal{H}$ respecting the way in which φ maps G to H. More formally, φ is a weak embedding if there is an embedding $\psi_\varphi : G \to \mathcal{H}$ that maps each vertex $v \in V$ to a point in $D_{\varphi(v)}$, and each edge $uv \in E(G)$ to a non-crossing curve with endpoints in $D_{\varphi(u)}$ and $D_{\varphi(v)}$ that passes through D_w (resp., R_{wx}) if and only if w (resp., wx) is contained in $\varphi(uv)$.

We say that φ is a *simplicial map* if it maps edges of G to pipes or clusters of H, i.e., no edge "passes through" a cluster. See Fig. 1.3(c)–(d). We denote by C_n the cycle with n vertices. We can convert an instance $\varphi : G \to H$ with $|V(G)| = n$ into a simplicial map $\varphi' : G' \to H$ with $|V(G')| = O(n^2)$ as follows. Subdivide every edge $e \in E(G)$ whose image $\varphi(e)$ contains a cluster $w \in V(H)$ at w. Then every edge in $E(G)$ corresponds to a path of length $O(n)$ in $E(G')$. The strategy of the algorithms in this thesis is to apply local operations to an instance producing a "simpler" instance until we obtain a base case that is easy to solve.

1.5 Scope

Chapter 2: Weakly simple polygons. In the case that G is a cycle, weak embeddings are equivalent to *weakly simple polygons*. Algorithms for recognizing weakly simple polygons were studied in [CEX15, CDBPP09]. The problem was first

shown to be solvable in $O(n^3)$ time [CDBPP09] when the input is a simplicial map, where n is the number of vertices. Using the subdivision technique in Section 1.4, the general problem becomes solvable in $O(n^6)$ time. Cheng et al. [CEX15] showed with a better analysis that the same algorithm terminates in $O(n \lg n)$ time for simplicial maps, and in $O(n^2 \lg n)$ time for the general case. They also give an $O(n)$-time algorithm for other restricted classes of inputs. Chapter 2 describes an algorithm that recognizes weakly simple polygons in $O(n \lg n)$ time in general. The result is obtained by using techniques from Arkin et al. [ABD+04] (designed to solve FOLDABILITY in 1D) and a set of new operations that carefully "untangle" sets of overlapping edges.

Chapter 3: Weak embeddings of graphs. Finding efficient algorithms for the recognition of weak embeddings $\varphi : G \to H$, where G is an arbitrary graph, was posed as an open problem in [AAET17, CEX15, CDBPP09]. The first polynomial-time solution for the general version follows from a recent variant [FK18] of the Hanani-Tutte theorem [Han34, Tut70], which was conjectured by M. Skopenkov [Sko03] in 2003 and in a slightly weaker form already by D. Repovš and A. Skopenkov [RS98] in 1998. However, this algorithm reduces the problem when φ is a simplicial map to a system of linear equations over $\mathbb{Z}_2$. The running time is dominated by solving this system in $O(m^{2\omega}) \leq O(m^{4.75})$ time, where $m = |E(G)|$, and $\omega < 2.373$ is the matrix multiplication exponent. This implies a running time of $O(m^{4\omega}) \leq O(m^{9.5})$ for the general case (non-simplicial maps). Chapter 3 describes an algorithm that recognizes whether a simplicial map is a weak embedding in $O(m \log m)$ time. This implies an $O(nm \lg n)$-time solution for the general case, where $n = |V(G)|$.

Chapter 4: NP-hardness for higher dimensions. In their seminal 1996 paper, Bern and Hayes initiated investigation into the computational complexity of origami [BH96]. They claimed that FOLDABILITY is NP-hard for 2D paper even if all faces of the paper are mapped to the same plane (FLAT-FOLDABILITY). A variant of the problem where the above-below relation between adjacent faces is given with the folded isometry (ASSIGNED FLAT-FOLDABILITY) is also claimed to be NP-hard. Since weak embeddings generalize FOLDABILITY, the results in [BH96] imply that it

is NP-hard to decide whether a given continuous piecewise linear map $\varphi : A \to \mathbb{R}^3$, where A is a simplicial 2-complex, is a weak embedding. Chapter 4 describes and corrects an error in Bern and Hayes' hardness proof, and provides an improvement in these hardness results by restricting the input to have faces bounded by lines at angles multiple of 45°. Let k denote the maximum number of overlapping faces at any given point of an instance. The new reduction also produces instances with $k = 9$ ($k = 25$ for the Assigned Flat-Foldability variant). This implies that the problem is not fixed-parameter tractable in k, i.e., there is no polynomial-time algorithm even for instances where $k = O(1)$ unless P = NP.

Chapter 5: Conclusion. This chapter gives an overview of future work and outstanding open problems.

Chapter 2

Weakly Simple Polygons

This chapter presents an $O(n \log n)$-time algorithm that determines whether a given n-gon in the plane is weakly simple. The result generalizes to recognizing weak embeddings of disjoint cycles and paths in $\mathbb{R}^2$. This improves upon an $O(n^2 \log n)$-time algorithm by Chang, Erickson, and Xu [CEX15]. Weakly simple polygons are required as input for several geometric algorithms. As such, recognizing simple or weakly simple polygons is a fundamental problem. The results in this chapter are joint work with Greg Aloupis, Jeff Erickson and Csaba Tóth [AAET17].

2.1 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 2.1).

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 [CEX15] for a detailed

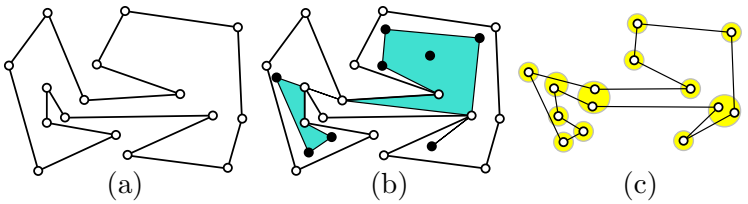


Figure 2.1: (a) A simple polygon P with 16 vertices. (b) Eight points in the interior of P (solid dots); their geodesic hull is a weakly simple polygon P' with 14 vertices. (c) A perturbation of P' into a simple polygon.

discussion and five equivalent definitions for weak simplicity of a polygon. Among others, a result by Ribó Mor [Mor06, 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 2.2). 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.) Then, the problem becomes equivalent to recognizing weak embeddings of a cycle in $\mathbb{R}^2$.

We can decide whether an n-gon in the plane is simple in $O(n \log n)$ time by a sweepline algorithm [SH76]. Chazelle's polygon triangulation algorithm also recognizes simple polygons (in $O(n)$ time), because it only produces a triangulation if the input is simple [Cha91]. Recognizing weakly simple polygons, however, is more subtle. Skopenkov [Sko03] gave a combinatorial characterization of the topological obstructions to weak simplicity in terms of line graphs. Cortese et al. [CDBPP09] gave an $O(n^6)$-time algorithm to recognize weakly simple n-gons. Chang et al. [CEX15] 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.) An instance is equivalent to a simplicial map if and only if it has no forks. 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 [CEX15, CDBPP09] and from Arkin et al. [ABD+04], and obtain the following main results. (Recall the definitions from Chapter 1.)

Theorem 2.1.1. *Given a polygon P with n vertices:*

1. *Deciding whether P is weakly simple can be done in $O(n \log n)$ time.*
2. *If P is weakly simple, a simple polygon with $2n$ vertices within Fréchet distance ε from P can be computed in $O(n \log n)$ time for every $\varepsilon > 0$.*

Theorem 2.1.2. *Given a continuous map $\varphi : G \to \mathbb{R}^2$ where G is a max-degree 2 graph, and $|V(G)| = n$ that maps every edge in $E(G)$ to a line segment:*

1. *Deciding whether φ is a weak embedding can be done in $O(n \log n)$ time.*
2. *If φ is a weak embedding, an embedding ψ_ε where $\|\varphi - \psi_\varepsilon\| < \varepsilon$ can be computed in $O(n \log n)$ time for every $\varepsilon > 0$.*

Recall that we can also phrase Theorem 2.1.2 in terms of a (non-simplicial) map $\varphi : G \to H$, where H is a plane graph. We first prove Theorem 2.1.1. Our decision algorithm is detailed in Sections 2.3–2.5. 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 reductions steps such as *crimp reductions* and *cluster expansions* (Section 2.3). Second, the *bar simplification* phase successively eliminates all forks (Section 2.4). Third, the *spur elimination* phase eliminates all spurs (Section 2.5). When neither forks nor spurs are present, we can decide weak simplicity in $O(n)$ time [CDBPP09]. 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 2.6). Finally, Section 2.8 generalizes the results for max-degree 2 graphs, proving Theorem 2.1.2.

2.2 Preliminaries

In this section, we review previously established definitions and known methods from [CEX15] and [CDBPP09].

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 [Grü12]. 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 [CDR02], [CDBPP09], and [Mor06, Theorem 3.1], an equivalent definition was formulated by Chang et al. [CEX15] 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 problem of recognizing weak embeddings of cycles and arcs has been considered [Min97, Sko03]. This problem is equivalent to recognizing weakly simple polygons and polygonal chains. We can then formulate the weak simplicity recognition problem by replacing G with a cycle in a instance $\varphi : G \to H$

of weak embedding recognition. A polygon P is then equivalent to the map φ.

Bar decomposition of H. Let H be a planar straight-line graph obtained from a geometric drawing φ of a graph G in $\mathbb{R}^2$ (edges are mapped to line segments). Recall that, to avoid confusion, we call vertices and edges of H *clusters* and *pipes* respectively (see Section 1.4). This graph can be computed in $O(n \log n)$ time by a sweep line algorithm [CEX15]. 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 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 Figure 2.2(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 [CEX15]. 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 pipes.

Every cluster in $V(H)$ that is not in the interior of a bar is called *sober*[1]. The set of clusters in H is $\{p_0, \ldots, p_{n-1}\}$ (note that P may have repeated vertices that correspond to the same cluster); two clusters are connected by a pipe in $E(H)$ if they are consecutive clusters along a bar; see Figure 2.2(b). Note that up to $O(n)$ edges of P may pass through a cluster of H, and there may be $O(n^2)$ edge-cluster pairs such that an edge of P passes through a cluster of H. 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, cluster expansion, bar expansion). We shall use these and introduce several new operations in Sections 2.3.3–2.5.

[1] We adopt terminology from [CEX15].

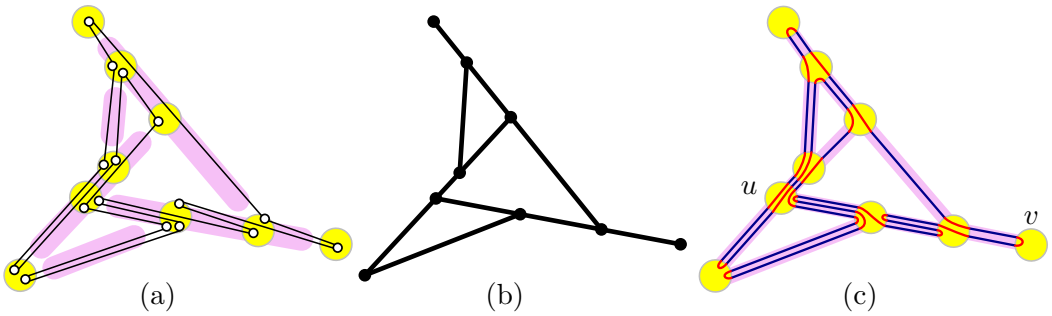


Figure 2.2: (a) The bar decomposition for a weakly simple polygon P with 16 vertices (P is perturbed into a simple polygon for clarity). (b) The image graph H of P. (c) A perturbation in a strip system $\mathcal{H}$ of H.

Combinatorial characterization of weak embeddings. To show that an operation is ws-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 weak embedding is perturbed into an embedding, overlapping edges in G are perturbed into interior-disjoint near-parallel edges, which define an ordering. It turns out that these orderings over all pipes of H are sufficient to encode an ε-perturbation and to (re)construct an ε-perturbation.

We rely on the notion of "strip system" introduced in [CEX15, Appendix B]. Similar concepts have previously been used in [CDR02, CDBPP09, FT17, Min97, Sko03]. Without loss of generality, we assume that no bar is vertical (so that the above-below relationship is defined between disjoint pipes parallel to a bar). The *strip system* $\mathcal{H}$ of H (a.k.a. thickening of H) is a 2-manifold with boundary constructed as follows: For every $u \in V(H)$, create a topological disk D_u, and for every edge $uv \in E(H)$, create a rectangle R_{uv}. For every D_u and R_{uv}, fix an arbitrary orientation of ∂D_u and ∂R_{uv}, respectively. Partition the boundary of ∂D_u into $\deg(u)$ arcs, and label them by $A_{u,v}$, for all $uv \in E(H)$, in the cyclic order around ∂D_u determined by the rotation of u in H. Finally, the manifold $\mathcal{H}$ is obtained by identifying two opposite sides of every rectangle R_{uv} with $A_{u,v}$ and $A_{v,u}$ via an orientation preserving homeomorphism (i.e., consistently with the chosen orientations of $\partial R_{uv}, \partial D_u$ and ∂D_v). See Fig. 2.1(b)–(c), where disks are shown in yellow and rectangles in purple. An embedding of G in $\mathcal{H}$ is *φ-respecting* if it maps each

vertex $v \in V$ to a point in the interior of $D_{\varphi(v)}$, and each edge $uv \in E(G)$ to a Jordan arc (i.e., a non-crossing arc) with endpoints in $D_{\varphi(u)}$ and $D_{\varphi(v)}$ that passes through D_w (resp., R_{wx}) if and only if w (resp., wx) is contained in $\varphi(uv)$. We denote by ψ_φ a φ-respecting embedding of G and say that ψ_φ *approximates* φ.

Given a φ-respecting embedding of G in $\mathcal{H}$, we can transform it into an embedding withing Fréchet distance ε from φ by deforming $\mathcal{H}$ (via homeomorphism) so that for every disk D_u it becomes a disk of radius ε centered at u and for every rectangle R_{uv} it contains only points ε^2 away from uv. There is a sufficiently small $\varepsilon_0 = \varepsilon_0(\varphi) > 0$, depending on φ, such that the disks D_u are pairwise disjoint, the rectangles R_{uv} are pairwise disjoint, and every rectangle R_{uv} of a pipe intersects only the disks at its endpoints D_u and D_v. These properties hold for all ε, $0 < \varepsilon < \varepsilon_0$.

Combinatorial representation by signatures. Let P be a polygon equivalent to a map $\varphi : G \to H$. A polygon is *in the strip-system* of H if its edges alternate between an edge that connects the boundary of two disks D_u and D_v and whose interior is contained in R_{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 rectangle R_{uv} form a perfect matching. See Figure 2.2(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 strip-system of H that are φ-respecting, i.e., cross the disks and rectangles in the same order as P traverses the corresponding clusters and pipes of H. By [CEX15, Theorem B.2], P is weakly simple if and only if $\Phi(P) \neq \emptyset$.

Let Q be a polygon in the strip system of H. For each pipe $uv \in E(H)$, the above-below relationship of the edges of Q in R_{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 pipes of H.

Given the signature $\sigma(Q)$ of a polygon Q in the strip system of H, we can easily (re)construct a simple polygon Q' with the same signature in the strip-system of H. For every pipe $uv \in E(H)$, let the *volume* $\mathrm{vol}(uv)$ be the number of edges of P that lie on uv. Place $\mathrm{vol}(uv)$ disjoint Jordan curves between ∂D_u and ∂D_v in

R_{uv} of the $\mathcal{H}$. Finally, for every disk D_u, construct a non-crossing 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. By construction, $Q \in \Phi(P)$ implies $Q' \in \Phi(P)$, since Q and Q' determine the same perfect matching between corresponding endpoints on ∂D_u at every node u, i.e., either both are φ-respecting or both are not.

Remark 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 H (i.e., the counterclockwise order of edges incident to each vertex), as long as H is a plane graph. Consequently, when an operation modifies the image graph, it is enough to maintain the combinatorial embedding of H (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 2.6, we introduce another combinatorial representation of $O(n)$ size that uses the ordering of the edges in each bar (rather than each pipe) of H.

Combinatorially different perturbations. In the absence of spurs, a polygon φ determines a unique noncrossing perfect matching in each disk D_u [CEX15, Section 3.3]. 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 2.3.

Figure 2.3: Two perturbations of a weakly simple polygon on 6 vertices (all of them spurs) that alternate between two distinct points in the plane.

2.3 Preprocessing

By a standard line sweep [SH76], we can detect and halt if any two edges properly cross. We then simplify the polygon, using some known steps from [ABD+04, CEX15], and some new ones. All of this takes $O(n \log n)$ time.

2.3.1 Crimp reduction

Arkin et al. [ABD+04] 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 denoted $[a, b, c, d]$ such that both the first edge ab and the last edge cd contain the middle edge bc (the containment need not be strict). The operation crimp-reduction(a, b, c, d) replaces the crimp $[a, b, c, d]$ with edge ad; see Figure 2.4.

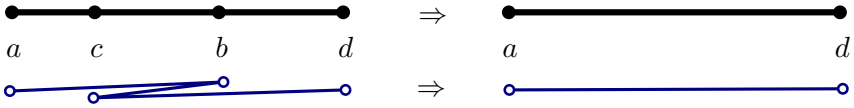


Figure 2.4: A crimp reduction replaces $[a, b, c, d]$ with ad. Top: H. Bottom: P.

Lemma 2.3.1. *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 rectangles 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 2.5. 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.

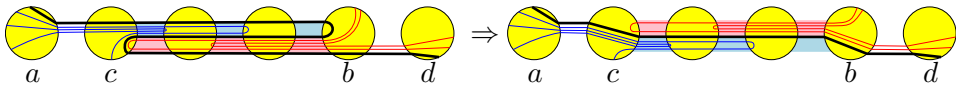


Figure 2.5: The operation crimp-reduction replaces a crimp $[a, b, c, d]$ with an edge $[ad]$.

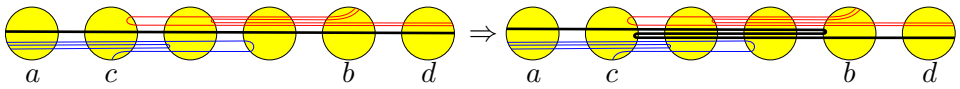


Figure 2.6: The reversal of crimp-reduction replaces edge $[ad]$ with a crimp $[a, b, c, d]$.

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 2.6. Replace edge $[a, d]$ by $[a, b, c, d]$ also replacing $[a, d]$ in the ordering of the affected pipes 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 [CEX15]. 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 2.3.2. *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. □

The operations introduced in Section 2.4 maintain properties (A1)–(A2) for all maximal paths inside an elliptical disk Δ_b.

2.3.2 Cluster expansion

Compute the bar decomposition of P and its image graph H (defined in Section 2.2, see Figure 2.2). For every sober cluster of H, we perform the ws-equivalent cluster-expansion operation, described in [CEX15, Section 3] (Cortese et al. [CDBPP09] also call this a cluster expansion, while Chang et al. call this a *node expansion*). Let u be a sober cluster of H. Let Δ_u be the disk centered at u with radius $\delta > 0$ sufficiently small so that Δ_u intersects only the pipes incident to u. For each pipe ux incident to u, create a new cluster u^x at the intersection point $ux \cap \partial\Delta_u$. Then modify P by replacing each subpath $[x, u, y]$ passing through u by $[x, u^x, u^y, y]$; see Figure 2.7. If a cluster expansion produces an edge crossing, report that P is not weakly simple.

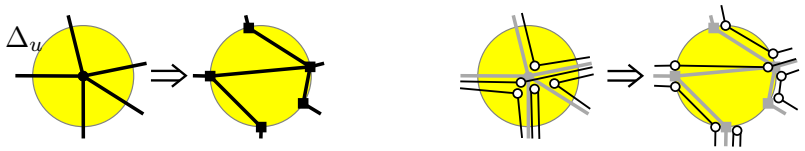


Figure 2.7: Cluster expansion. (Left) Changes in H. (Right) Changes in P (the vertices are perturbed for clarity). New clusters are shown as squares.

2.3.3 Bar expansion

Chang et al. [CEX15, Section 4] define a bar expansion operation. In this chapter, we refer to it as old-bar-expansion. For a bar b of H, draw a long and narrow ellipse Δ_b around the interior clusters of b, create subdivision vertices at the intersection of $\partial\Delta_b$ with the edges, and replace each maximal path in Δ_b by a straight-line edge. If b contains no spurs, old-bar-expansion is known to be ws-equivalent [CEX15]. Otherwise, it can produce false positives, hence it is not ws-equivalent; see Figure 2.8 for an example.

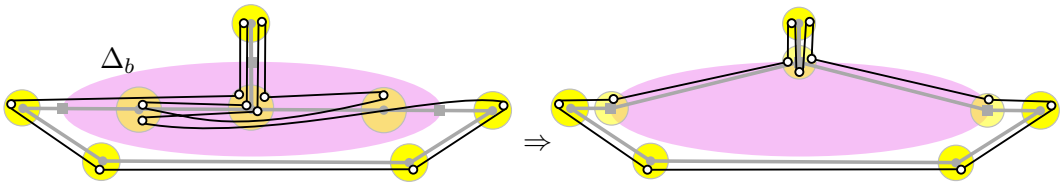


Figure 2.8: The old-bar-expansion converts a non-weakly simple polygon to a weakly simple one.

New bar expansion operation. Let b be a bar of H with at least one interior cluster; see Figure 2.9. Without loss of generality, assume that b is horizontal. Let Δ_b be an ellipse whose major axis is in b such that Δ_b contains all interior clusters of b (clusters in b except its endpoints), but does not contain any other cluster of H and does not intersect any pipe that is not incident to some cluster inside Δ_b.

Similar to old-bar-expansion, the operation new-bar-expansion introduces subdivision vertices on $\partial\Delta_b$, however we keep all interior vertices of a bar at their original positions. In Section 2.4, we apply a sequence of new operations to eliminate all vertices on b sequentially while creating new clusters in the vicinity of Δ_b. Our bar expansion operation can be considered as a preprocessing step for this

subroutine.

For each pipe ux between a cluster $u \in b \cap \Delta_b$ and a cluster $x \notin b$, create a new cluster u^x at the intersection point $ux \cap \partial\Delta_b$ and subdivide every edge $[u, x]$ to a path $[u, u^x, x]$. For each endpoint v of b, create two new clusters, v' and v'', as follows. Cluster v is adjacent to a unique pipe $vw \subset b$, where $w \in b \cap \Delta_b$. Create a new cluster $v' \in \partial\Delta_b$ sufficiently close to the intersection point $vw \cap \partial\Delta_b$, but strictly above b; and create a new cluster v'' in the interior of pipe $vw \cap \Delta_b$. 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-pipes of the bars), it is ws-equivalent.

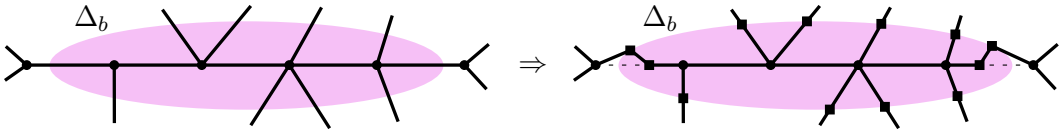


Figure 2.9: The changes in H caused by new-bar-expansion.

Crossing paths. Apart from cluster-expansion and old-bar-expansion, none of our operations creates edge crossings. In some cases, our bar simplification algorithm (Section 2.4) detects whether two subpaths cross. Crossings between overlapping paths are not easy to identify (see [CEX15, Section 2] for a discussion). We rely on the following simple condition to detect some (but not all) crossings.

Lemma 2.3.3. *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 2.10.) 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. □

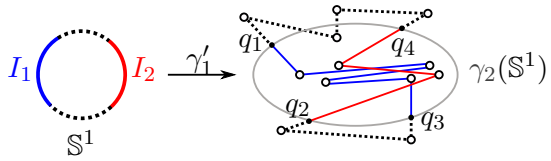


Figure 2.10: Forbidden configuration described by Lemma 2.3.3.

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

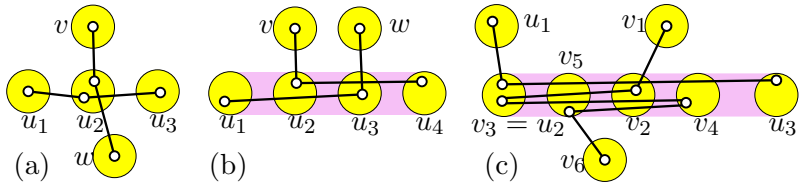


Figure 2.11: Three pairs of incompatible paths.

Corollary 2.3.4. *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 distinct pipes in this cyclic order around* u_2 *(*cluster crossing*; see Figure 2.11(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 clusters* v *and* w *lie in an open halfplane bounded by this line (Figure 2.11(b)).*
3. $[u_1, u_2, u_3]$ *and* $[v_1, v_2, \ldots, v_{k-1}, v_k]$ *where* $v_2 \in \text{int}(u_2u_3)$*,* $v_3, \ldots, v_{k-1} \in \{u_2\} \cup \text{int}(u_2u_3)$*, clusters* u_1 *and* v_1 *lie in an open halfplane bounded by the supporting line of* u_2u_3*, and cluster* v_k *lies on the other open halfplane bounded by this line (Figure 2.11(c)).*

Proof. In all four cases, Lemma 2.3.3 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 pipe 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 Δ_b. All clusters $u \in \partial\Delta_b$ lie either above or below b. We call them *top* and *bottom* clusters, respectively. Let $\mathcal{P}$ denote the set of maximal paths $p = [u_1^x, u_1, \ldots, u_k, u_k^y]$ in Δ_b. The paths in $\mathcal{P}$ are classified based on the position of their endpoints. A path p can be labeled as follows:

- *cross-chain* if u_1^x and u_k^y are top and bottom clusters respectively,
- *top chain* (resp., *bottom chain*) if both u_1^x and u_k^y are top clusters (resp., bottom clusters),
- *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.

2.3.4 Clumps

As a preprocessing step for spur elimination (Section 2.5), we group all clusters that do not lie inside a bar into *clumps*. After cluster-expansion and new-bar-expansion, all such clusters lie on a boundary of a disk (circular or elliptical). For every sober cluster u, we create $\deg(u)$ clumps as follows. Refer to Figure 2.12. The cluster expansion has replaced u with new clusters on $\partial\Delta_u$. Subdivide each pipe in Δ_u with two new clusters. For each cluster $v \in \partial\Delta_u$, form a clump $C(v)$ that consists of v and all adjacent (subdivision) clusters inside Δ_u. For each cluster u on the boundary of an elliptical disk Δ_b, subdivide the unique edge outside Δ_b incident to u with a cluster u^*. Form a clump $C(u^*)$ containing u and u^*. Every clump maintains the following invariants.

Clump Invariants. For every clump $C(u)$:

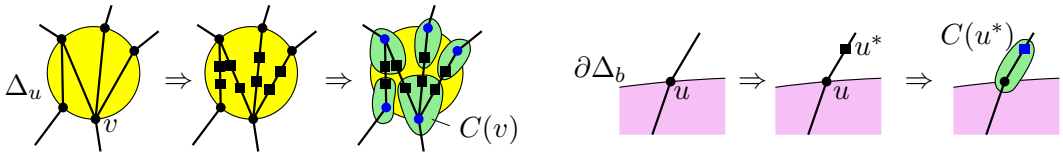


Figure 2.12: Formation of new clumps around (left) a sober cluster and (right) a cluster on the boundary of an elliptical disk. The roots of the induced trees are colored blue.

(I1) $C(u)$ induces a tree $T[u]$ in H 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 cluster ℓ satisfies one of the following conditions:

  (a) ℓ has degree one in H (and every vertex at ℓ is a spur);

  (b) ℓ has degree two in H and there is no spur at ℓ.

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

Dummy vertices. Although the operations described in Sections 2.4 and 2.5 introduce new clusters in the clumps, the image graph H will always have $O(n)$ clusters and pipes. A vertex at a clump cluster is called a *benchmark* if it is a spur or if it is at a leaf cluster; otherwise it is called a *dummy vertex*. Paths traversing clumps may jointly contain $\Theta(n^2)$ dummy vertices in the worst case, however we do not store them explicitly. By (I1), (I2), and (I3) a maximal path in a clump can be uniquely encoded by one benchmark vertex: if it goes from a root to a spur at an interior cluster s and back, we record only $[s]$; and if it traverses $T[u]$ from the root to a leaf ℓ, we record only $[\ell]$.

2.4 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 2.4.1), define them and show that they are ws-equivalent (Sections 2.4.2–2.4.3), and then show how to use these operations to eliminate all vertices from a bar (Section 2.4.4).

2.4.1 Overview

After preprocessing in Section 2.3, we may assume that P has no edge crossings and satisfies (A1)–(A2). Further, we also assume that properties (A1)–(A2) are restored, if necessary, after every operation defined in this section by suitable merge and crimp-reduction operations. We can check in $O(1)$ time whether an edge created by an operation is collinear with adjacent edges or part of a crimp, and apply the operation in $O(1)$ time. Thus the cost of maintaining (A1)–(A2) is absorbed by the creation of the edges involved.

We summarize the overall effect of the bar simplification subroutine for a given expanded bar.

Changes in H. Refer to Figure 2.13. All clusters in the interior of the ellipse Δ_b are eliminated. Some spurs on b are moved to new clusters in the clumps along $\partial\Delta_b$. Pipes inside Δ_b connect two leaves of trees induced by clumps.

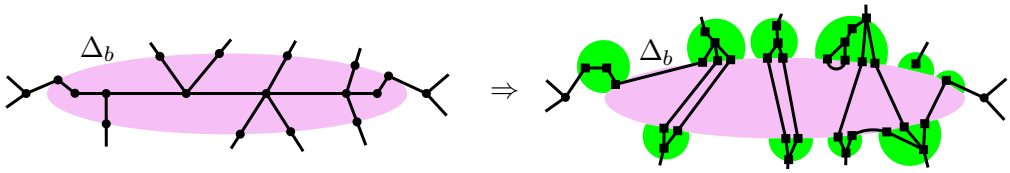


Figure 2.13: The changes in H caused by a bar simplification.

Changes in the polygon P. Refer to Figure 2.14. Consider a maximal path p in P that lies in Δ_b. The bar simplification replaces $p = [u, \ldots, v]$ with a new path p'.

By (I3)-(I4), only clusters u and v in p lie on $\partial\Delta_b$. 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 clump containing u (Figure 2.14 (a)). If p has no such decomposition, but its two endpoints are at the same cluster, $u = v$, then p' is a single edge connecting two leaves in the clump containing u (Figure 2.14 (b)). If the endpoints of p are at two different clusters, p' is an edge between two leaves of the clumps containing u and v respectively (Figure 2.14 (c) and (d)).

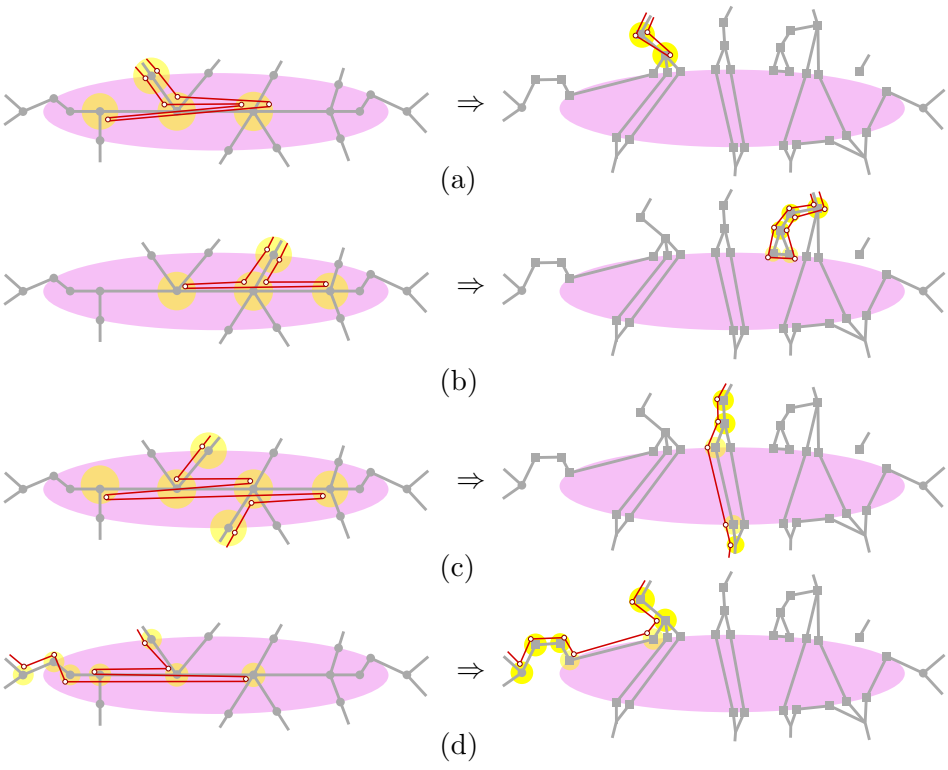


Figure 2.14: The changes in the polygon caused by a bar simplification.

2.4.2 Primitives

The operations in Section 2.4.3 rely on two basic steps, spur-reduction and cluster-split (see Figure 2.15). Together with merge and subdivision, these operations are called *primitives*.

spur-reduction(u, v)**.** Assume that every vertex at cluster 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 2.15, left.)

cluster-split(u, v, w)**.** Assume that pipes uv and vw are consecutive in radial order around v, cluster 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 cluster 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 2.15, right.)

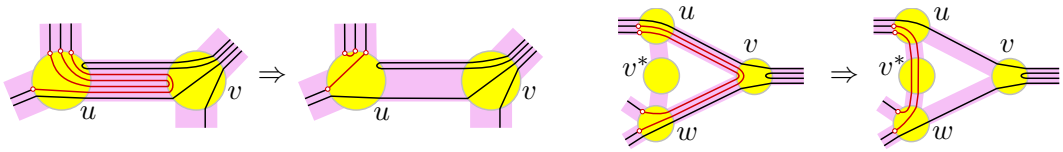


Figure 2.15: Left: Spur-reduction(u, v). Right: Cluster-split(u, v, w).

The following two lemmas are generalizations of the results in [CEX15, Section 5].

Lemma 2.4.1. *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 R_{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 occurrences of vertex $[u]$ 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 chord of u_2 from R_{uv} within the disk D_u. Intuitively, we have $u_1 \prec u_2$ if

u_1 blocks u_2 from the rectangles R_{uv}. Note that if $u_1 \prec u_2$, then neither endpoint of the chords corresponding to u_1 is on the boundary of R_{uv}; consequently u_1 was obtained from a path $[u, v, u]$ or $[u, v, u, v, u, \ldots, u]$ 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 $[p, u, v, u, v, u, \ldots, u, q]$ 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 R_{uv}; consequently, none of them blocks access to $R_{u,v}$. 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 2.4.2. *Operation* cluster-split *is ws-equivalent.*

Proof. Let P' be obtained from P via cluster-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 2.15(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 pipes 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 H 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. □

2.4.3 Operations

We describe three more complex operations: pin-extraction, V-shortcut, and L-shortcut. In Section 2.4.4, 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 Δ_b are simplified incrementally, removing one vertex at a time, by the L-shortcut operation.

Since H 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 clusters and pipes in H, and use them to describe the changes in the polygon. In the last step of these operations, we remove any cluster (pipe) that contains no vertex (edge), to ensure that H 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), cluster v is adjacent to a unique cluster w outside of Δ_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 H. See Figure 2.16 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 pipe uq is in the interior of the wedge $\angle v_1 u v_2$. By (I3), clusters v_1 and v_2 are each adjacent to unique clusters w_1 and w_2 outside of Δ_b, respectively.

The operation executes the following primitives sequentially: (1) cluster-split(v_1, u, v_2), which creates a temporary cluster u^*; (2) cluster-split(u^*, v_1, w_1) and cluster-split(u^*, v_2, w_2); which create $v_1^*, v_2^* \in \partial\Delta_b$, respectively; (3)

merge every path $[v_1^*, u^*, v_2^*]$ to $[v_1^*, v_2^*]$. (4) Update the H. See Figure 2.17 for an example.

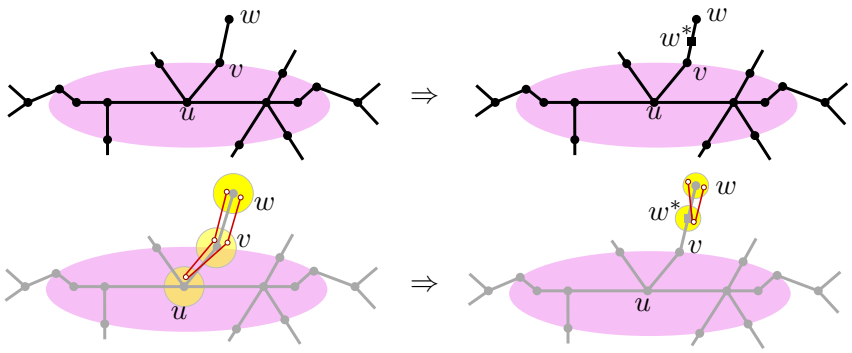


Figure 2.16: pin-extraction. Changes in H (top), changes in the polygon (bottom).

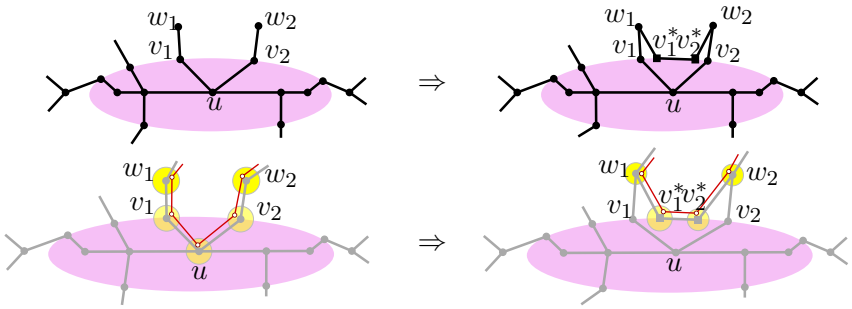


Figure 2.17: V-shortcut. Changes in H (top), changes in the polygon (bottom).

Lemma 2.4.3. pin-extraction *and* V-shortcut *are ws-equivalent and maintain (I1)–(I4).*

Proof. pin-extraction**.** By construction, the operation maintains (I1)–(I4). 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 (I1)–(I4). The first two primitives are ws-equivalent by Lemma 2.4.2. The third step is ws-equivalent because triangle $\Delta(u^* v_1^* v_2^*)$ is empty of clusters and pipes, by assumption. □

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 2.18. For a cluster $v \in \partial\Delta_b$, let L_v be the

set of paths $[v, u_1, u_2]$ in P such that $u_1, u_2 \in \text{int}(\Delta_b)$. 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}.

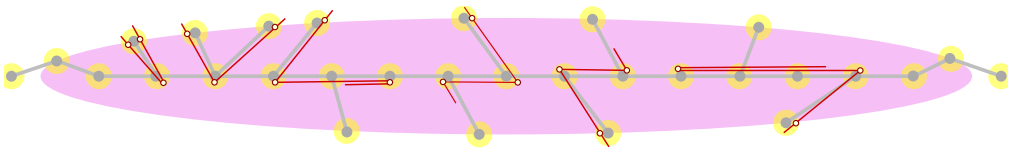


Figure 2.18: Paths in $\mathcal{P}in$, $\mathcal{V}$, L_v^{TR}, L_v^{TL}, L_v^{BR}, and L_v^{BL}.

We partition L_v into four subsets (refer to Figure 2.18): 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), $v \in \partial\Delta_b$ and $L_v^{TR} \neq \emptyset$. By (I3), v is adjacent to a unique cluster $u_1 \in b$ and to a unique cluster $w \notin \Delta_b$. Let U denote the set of all clusters 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 cluster 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 pipe pu_1 is in the interior of the wedge $\angle v u_1 u_{\min}$;

(B3) no edge $[p, q]$ such that $p \in \partial\Delta_b$ is a top vertex and $q \in b$, $x(u_1) < x(q) < x(u_{\max})$.

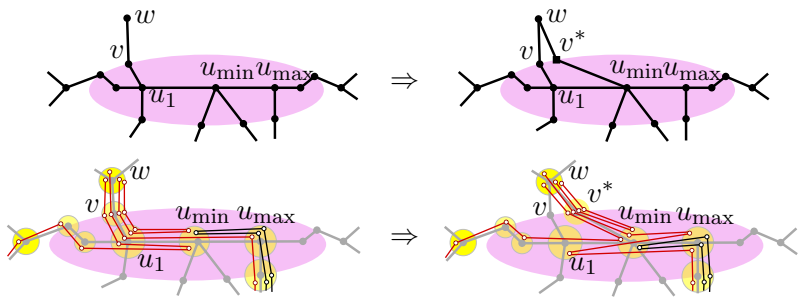


Figure 2.19: L-shortcut. Changes in H (top), changes in the polygon (bottom).

Do the following (see Figure 2.19 for an example).

(0) Create a new cluster $v^* \in \partial\Delta_b$ 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 $u_2 \in U$ and $u_2 \neq u_{\min}$, by $[w, v^*, u_{\min}, u_2]$.

(4) Update H.

See Figure 2.20 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 2.4.4. L-shortcut *is ws-equivalent and maintains (I1)–(I4) and (A1)–(A2).*

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 2.3.1. Let W 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, cluster-split, and merge. Specifically, they are equivalent to subdividing every edge in W into $[u_1, u_{\min}, u_2]$ whenever $u_2 \neq u_{\min}$, and applying cluster-split$(v, u_1, u_{\min})$ (which creates u_1^*) to P_2 followed

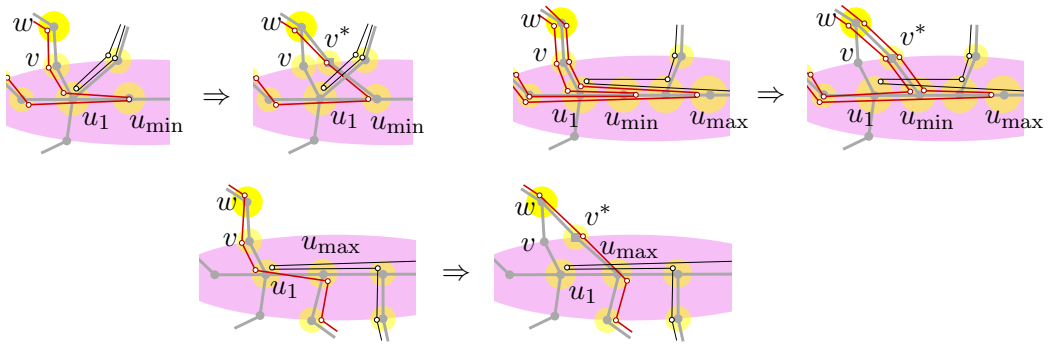


Figure 2.20: Cases in which L-shortcut is not ws-equivalent. Top left: P has a pin $[v, u_1, v]$ not satisfying (B2). Top right: P does not satisfy (B3). Bottom: the operation skips phase (1).

by cluster-split(w, v, u_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 cluster-split$(v, u_1, u_{\min})$: pipe $u_1 u_{\min}$ may be collinear with several pipes of b, and P_2 may contain spurs that overlap with $u_1 u_{\min}$. In the next paragraph, we show that the spurs that may overlap with $u_1 u_{\min}$ do not pose a problem, and we can essentially repeat the proof of Lemma 2.4.2.

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 the form $[v, u_1, u_2, u_3]$ 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 $[u_1, u_2] \in W$ lies above all overlapping edges $e \notin W$.

Let $Q_1 \in \Phi(P_1)$ be a perturbation of P_1 into a simple polygon that has the minimum number of edges $[u_1, u_2] \in W$ 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 2.21. Recall that Q_1 implies a total order of edges in each pipe of b based on the above-below relationship. Let $[u_1, u_2'] \in W$ 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 rectangles between u_1 and u_2', and are not in W. By (B2)–(B3) and Lemma 2.3.2, 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 pipes 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 W above $[u_1, u_2']$ satisfy $(\star)$, contradicting the choice of Q_1.

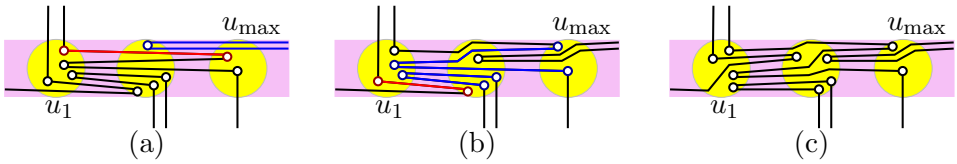


Figure 2.21: (a) A perturbation Q_1 that violates property $(\star)$; the highest edge $[u_1, u_2'] \in W$ that violates $(\star)$ is red, and edges in Z are blue. (b) We can modify Q_1 to reduce the number of edges in W that violate $(\star)$. (c) There exists a perturbation Q_1 that satisfies $(\star)$.

We can proceed as in the proof of Lemma 2.4.2, 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 Δ_b. □

2.4.4 Bar simplification algorithm

In this section, we describe an algorithm called bar-simplification to incrementally remove 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 Δ_b contains edges along bar b, and the sequence of these edge lengths is unimodal (cf. Lemma 2.3.2). 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 2.22). 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 Δ_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 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.

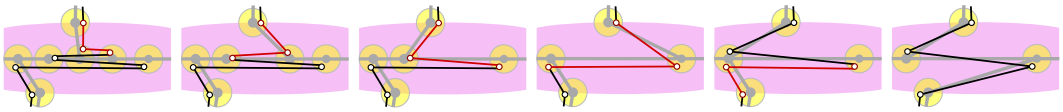


Figure 2.22: Life cycle of a cross-chain in the while loop of bar-simplification. The steps applied, from left to right, are: (iii), (iv), (iii), (iv), and (vi).

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 pipe 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 $v \in \partial\Delta_b$ such that $\widehat{L}_v^{TR} \neq \emptyset$, do:

  (a) Let v be the rightmost cluster 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 cluster 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 $v \in \partial\Delta_b$ 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 $v \in \partial\Delta_b$ 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 $v \in \partial\Delta_b$ 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 2.4.5. *The operations performed by* bar-simplification(P, b) *are ws-equivalent, and maintain properties (A1)–(A2) and (I1)–(I4) inside Δ_b. The algorithm either removes all clusters from the ellipse Δ_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 pipe uq is in the interior of the wedge $\angle v_1uv_2$ (or else H would have a crossing). If such an edge exists, it is part of a path $[p, u, q]$. The cluster q is in $\partial\Delta_b$ between v_1 and v_2. Note that $p \neq q$, otherwise $[p, u, q]$ would be a pin. Further, p cannot be a cluster in the interior of the wedge $\angle v_1uv_2$, 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 $\angle v_1uv_2$. In this case, the paths $[v_1, u, v_2]$ and $[p, u, q]$ form the forbidden

configuration in Corollary 2.3.4(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 2.4.3. 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 2.4.4. 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 $p \in \partial\Delta_b$ is a top cluster. Edge $[p, q]$ is part of some path $[p, q, r]$. Note that r cannot be a top vertex of $\partial\Delta_b$, 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 cluster 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 2.3.4(1) or (3).

Assume now that v' exists but $L_{v'}^{TL}$ does not satisfy (B2) or (B3). 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 2.3.4(2). Else, we have $x(u_1) \leq x(u'_{\max}) < x(u'_1) < x(u_{\max})$. This implies that any edge $[p, q]$ that violates (B2) or (B3) for $L_{v'}^{TL}$ must also violate (B2) or (B3) for L_v^{TR}. However, this contradicts the choice of v (rightmost where $L_v^{TR} \neq \emptyset$) and v' (leftmost, $x(v) < x(v')$, where $L_{v'}^{TL} \neq \emptyset$).

Next assume that there is a path $[v', u'_1, u'_2] \in L_{v'}^{TL}$ 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 pipe $u'_1u'_2$, and p' is a bottom vertex. Now $[v, u_1, u_{\max}]$ and this cross-chain form the forbidden configuration in Corollary 2.3.4(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 2.4.4.

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 $\widehat{L}^{BL}_v \cup L^{BR}_v$. If the while loop terminates, every edge in b is necessarily in M_{cr} and $L^{TL}_v \cup L^{BR}_v = \emptyset$. Consequently, by Lemma 2.3.2, b contains no spurs and old-bar-expansion is ws-equivalent. This operation eliminates all clusters in the interior of Δ_b.

Termination. Each pin-extraction and V-shortcut operation reduces the number of vertices of P within Δ_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} does not contain any edge in M_{cr}, and L-shortcut does not create crimps. We may assume that step (iii) created 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 clusters 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 $v \in \partial\Delta_b$ to a nearby new cluster $v^* \in \partial D$. 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.

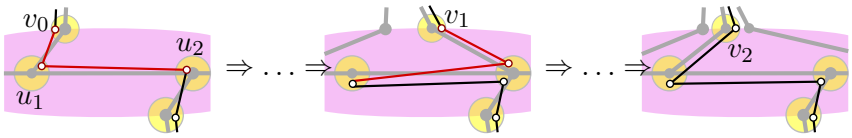


Figure 2.23: Only a single crimp can be created in a cross-chain by step (iii)b.

Suppose, for the sake of 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 2.23). 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 $\widehat{L}_v^{TR} = \emptyset$ for all top clusters v, and $L_{v'}^{TL} = \emptyset$ for all top clusters 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 cluster is at or to the right of the top cluster 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 $\widehat{L}_{v'}^{TR} = L_{v'}^{TL} = \emptyset$ for all top clusters v', $x(v') < x(v_2)$.

When L-shortcut(v_2, TR) is performed, we have $[v_2, u_1, u_2] \in L_{v_2}^{TR}$ but $[v_2, u_1, u_2] \notin \widehat{L}_{v_2}^{TR}$ (since u_1u_2 is the longest edge of c). Step (iii) is performed only if $\widehat{L}_p^{TR} \neq \emptyset$ for some top vertex p. Since the rightmost top vertex where $L_v^{TR} \neq \emptyset$ 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 $[v_2, u_1, u_3] \in L_{v_2}^{TL}$, such that $[u_1, u_3]$ is not the longest edge of c'. Since $L_{v_1}^{TR}$ contains the first two edges of both c and c', they have been modified by the same sequence of L-shortcut operations. Consequently, c' contained $[v, u_1, u_2, u_1]$ initially. By Lemma 2.3.2, 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 $\widehat{L}_{v_2}^{TR} = \emptyset$, contradicting the condition

in Step (iii). □

Lemma 2.4.6. *Algorithm* bar-simplification(P, b) *takes* $O(m \log m)$ *time, where* m *is the number of vertices in* b.

Proof. Operations pin-extraction, V-shortcut, and L-shortcut each make $O(1)$ changes in H. Operations pin-extraction and V-shortcut decrease the number of vertices inside Δ_b. Each L-shortcut does as well, except for the steps that create crimps. By Lemma 2.4.4, L-shortcut operations may create at most $2|\mathcal{P}| = O(m)$ crimps. So the total number of operations is $O(m)$.

When $[v, u_1, u_2] \in L_v^{TR}$ 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 $\Theta(m^2)$.

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 $v \in \partial\Delta_b$ 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 clusters in the paths $[v, u_1, u_2] \in L_v^X$ 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 $[v, u_1, u_2] \in L_v^X$ whether it is part of a cross-chain, and whether $[u_1, u_2]$ is the only longest edge of that chain.

The condition in step (ii) can be tested in $O(1)$ time by checking whether uv_1 and uv_2 are consecutive pipes in the rotation of cluster u in $V(H)$. 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 one of steps (iii)–(iv) which can be generalized to other occurrences of the L-shortcut operation. Conditions (B2)–(B3) can be checked in constant time by checking the circular order of edges incident to a cluster in $\partial\Delta_b$. 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 $u_3 \in \partial\Delta_b$

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 $[v, u_1, u_2] \in L^{TR}_v$, 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 Δ_b, which is $O(m)$. Each insertion and deletion takes $O(\log m)$ time, and the overall time complexity is $O(m \log m)$. □

2.5 Spur elimination algorithm

After bar-simplification (Section 2.4), we obtain a polygon that has no forks and every spur is at an interior cluster of some clump (formed on the boundary of some ellipse Δ_b). In the absence of forks, we can decide weak simplicity using [CEX15, 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 clumps with adjacent roots together. Recall that by (I1) each clump induces a tree. We also modify H, transforming each tree into a binary tree using ws-equivalent primitives. For each cluster s with more than two children, let s_1 and s_2 be the first two children in counterclockwise order. Create new clusters s'_1 and s'_2 by subdivision in ss_1 and ss_2, respectively, and create a pipe $s'_1 s_2$. Use the inverse of cluster-split to merge clusters s'_1 and s'_2 into a cluster 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 pipe uv connects clusters in different clumps, both u and v are leaves or both are root clusters. For every pair of clumps, $C(u)$ and $C(v)$, with adjacent roots, u and v, create a *group*

$G_{uv} = C(u) \cup C(v)$; see Figure 2.24. By construction, the groups are pairwise disjoint. Two groups are called *adjacent* if they have two adjacent leaves in H.

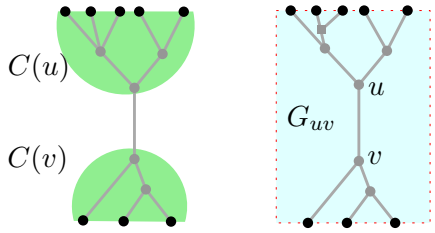


Figure 2.24: The formation of a group G_{uv}, containing clumps $C(u)$ and $C(v)$. Leaf clusters are shown as black dots.

Recall that a maximal path in each clump 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 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 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 clusters of H into groups satisfying (I1)–(I4). We consider one group at a time, and eliminate all spurs from one clump 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 2.4). 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 clump $C(u)$, and create its supporting data structures (described in Section 2.5.1 below).
2. While $T[u]$ contains an interior cluster, do:
   (a) If u contains no spurs and is incident to only two edges uv and uw, eliminate u with a merge operation. Rename cluster w to u which becomes the new root of the tree $T[u]$.
   (b) If u contains spurs, eliminate them as described in Section 2.5.2.
   (c) If u contains no spurs, split G_{uv} into two groups along a chain of pipes that contains uv as described in Section 2.5.3. Rename a largest resulting group to G_{uv}.

The detailed description of steps 2b and 2c are in Sections 2.5.2 and 2.5.3, respectively. We first present supporting data structures in Section 2.5.1, and then analyze the algorithm in Section 2.5.4.

2.5.1 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 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 $O(|\mathcal{B}|)$ clusters that contain benchmark vertices or have degree higher than 2. Dummy clusters 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 clusters of degree two cannot always be replaced by a straight-line segment (this might introduce unnecessary crossings). By Remark 2.2, it suffices to maintain the combinatorial embeddings of the trees $T[u]$ and $T[v]$ (i.e., the counterclockwise order of the incident pipes around

each cluster).

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 clusters 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 clusters in N, and then merge all clusters of degree two that are not in N. In particular, the clusters of $T(\mathcal{B}')$ include N and the lowest common ancestor of any two clusters in $N \cap C(u)$ and in $N \cap C(v)$, respectively. Denote by $\mathrm{lca}(r, s)$ the *lowest common ancestor* of clusters r and s in $T[u]$ (resp., $T[v]$).

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

- We store trees $T[u]$ and $T[v]$ each using the dynamic data structure of [CH05], which supports $O(1)$-time insertion and deletion of leaves, merging interior clusters 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 2.25(a)). For each tree, we maintain a left-to-right Euler tour in an order-maintenance data structure [BCD$^+$02, DS87], which supports insertions immediately before or after an existing item, deletions, and precedence queries, each in $O(1)$ amortized time. For any cluster 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-

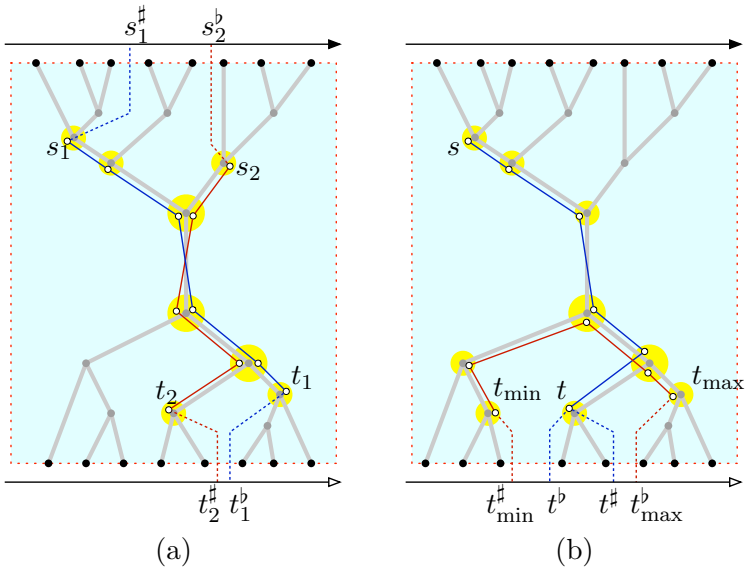


Figure 2.25: The geometry of crossing benchmark-to-benchmark paths. (a) Paths $[s_1; t_1]$ and $[s_2; t_2]$ cross. (b) If $t^\sharp_{\min} < t^\flat \leq t^\sharp < t^\flat_{\max}$, then any benchmark-to-benchmark paths $[s; t]$ crosses path $[t_{\min}; t_{\max}]$.

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 cluster 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.
- We use a central *interval tree* [dBvKOS00] for all $O(n)$ intervals $I[s_1; t; s_2]$ that can report, for a query cluster 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 2.25(a)).

Lemma 2.5.1. *Let s_1 and s_2 be arbitrary clusters in tree $T[u]$, and let t_1 and t_2 be arbitrary clusters 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) $s_1^\flat > s_1^\sharp$ and $t_1^\sharp > t_2^\flat$.*

2.5.2 Eliminating spurs from a root

We describe step 2b of Algorithm *spur-elimination*. Suppose that the root cluster u contains a spur. The following operation eliminates all spurs from u, but the resulting clump $C(v)$ need not satisfy (I2) and (I3), and we need to perform other operations to restore these properties. Refer to Figure 2.26(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 2.5.2. 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(u, v). In a BFS traversal of all clusters 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 cluster x has an edge in the pipe $x\,\text{parent}(x)$, 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 2.5) 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 applies for 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 cluster 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 H, 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 H, then do: by (I3), cluster t_1 is adjacent to a unique cluster $z \notin G_{uv}$. Subdivide pipe $t_1 z$ with a new cluster z^* (added to the clump containing z), subdivide every edge $[t_1, z]$ into $[t_1, z^*, z]$, and then replace every path $[z, z^*, t_1, z^*, z]$ with $[z]$. See Figure 2.26(b)–(c) for an example.

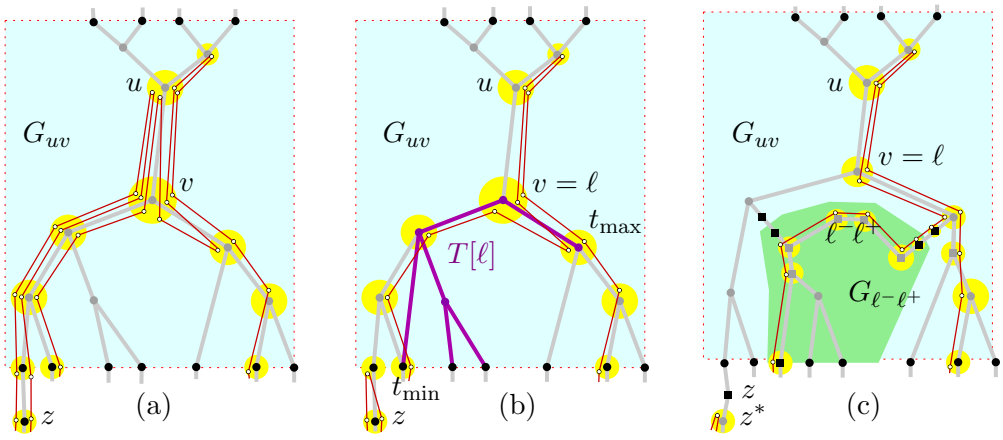


Figure 2.26: (a) Cluster u contains spurs. (b) After eliminating spurs, $T[v]$ does not satisfy (I2). (c) The analogues of pin-extraction and V-shortcut.

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(v).

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 cluster 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 cluster 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 2.26).

Let N^- (resp., N^+) be the set of clusters 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 clusters $t_1 \in N^-$ by $t_1^\sharp$, and let $t_{\min}$ be the minimum cluster; and similarly sort the clusters $t_2 \in N^+$ by $t_2^\flat$, and let $t_{\max}$ be the maximum cluster. The following lemma shows that interior clusters of the path from $t_{\min}$ to ℓ in $T[v]$ have no right branches, and the interior clusters of the path from $t_{\max}$ to ℓ have no left branches.

Lemma 2.5.3. *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 2.25(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 cluster $t^* \in C$, 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 clusters t such that $t_{\min}^\sharp < t^\flat \leq t^\sharp < t_{\max}^\flat$, in $O(|N'| + \log n)$ time, by binary searching in the list of leaves to find the leaves between $t_{\min}$ and $t_{\max}$ and using lowest common ancestor queries to find clusters in N'. The algorithm reports that the input polygon is not weakly simple and halts if some cluster in N' has a path satisfying Lemma 2.5.3. We can now assume that $N' \subset N$. The clusters in N induce a binary tree, denoted $T[\ell]$, of size at most $2|N|$: its clusters are all clusters in N and the lowest common ancestors of consecutive clusters in N^- and N^+ respectively. Note that a pipe of $T[\ell]$ might not correspond to a pipe of $T[v]$ (see Figure 2.26(b)). Denote by C^* the path between $t_{\min}$ and $t_{\max}$ in $T[\ell]$.

We now define the changes in H. Every cluster $t \in N \backslash C^*$ is deleted from G_{uv}, and added to the new group. Create two clusters, 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 pipe $\ell^-\ell^+$. Duplicate each cluster $t \in C^* \setminus \{\ell\}$, by creating a cluster t' (added to $G_{\ell^-\ell^+}$) sufficiently close to t, and add a pipe tt'. Subdivide every pipe tt' with two new boundary clusters, t_{leaf} (added to $T[v]$) and t'_{leaf} (added to $G_{\ell^-\ell^+}$). The clusters t or t' might now have degree 4. Adjust H so that the group trees are binary. Finally partition the clusters 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 $[t; t_2] \in \mathcal{S}_\ell$; and by $[t_{\text{leaf}}, t'_{\text{leaf}}, t_1]$ otherwise. 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 cluster-split at ℓ, a sequence of *clustersplits* along the chain C from ℓ to $t_{\min}$ and $t_{\max}$, respectively, subdivision operations that create the new leaf clusters, and merge operations at degree two clusters 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.

2.5.3 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 2.27.

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 cluster that we split into two nonempty clusters. 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.

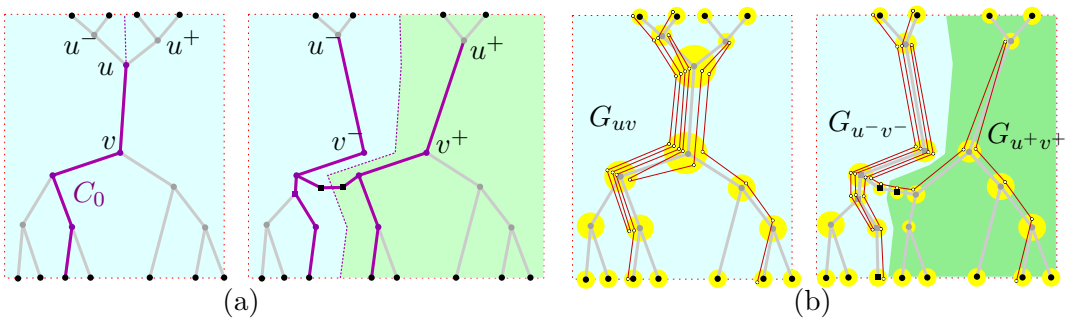


Figure 2.27: Splitting group G_{uv}. (a) Changes in H. (b) Changes in the polygon.

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 2.5.1, 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 $\mathcal{B}_{\ell_0} \cap \mathcal{B}^- \neq \emptyset$, 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 $O(|\mathcal{B}^-|)$ clusters, but we can test whether a query

cluster t of $T[v]$ is in C_0 in $O(1)$ time by checking whether $\text{lca}(\ell_0, t) = t$. Since the paths in $\mathcal{B}^-$ and $\mathcal{B}^+$ do not cross, all clusters of $T[v^-]$ are in or to the left of the chain C_0, and all common clusters of $T[v^-]$ and $T[v^+]$ are in C_0. The image graph of $T[v^-]$ (subgraph of H) can be computed from scratch using $\mathcal{B}^-$ in $O(\mathcal{B}^-)$ time. Replace each cluster 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 cluster deletion and merge operations as follows. First delete all clusters that are in $T[v^-]$ but not in C_0. For every cluster t of $T[v^-]$ that lies in C_0, if t has degree two in $T[v^-]$ and $\mathcal{B}_t^+ = \emptyset$, then it is a degree two cluster in $T[v]$ with no spurs, and so we can delete t by merging its two incident pipes. As a result, $T[v]$ 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 clusters t such that $[s_1; t; s_2] \in \mathcal{V}^0$. Each cluster $t \in N^0$ is in C_0 and, therefore, has a copy t^- in $T[v^-]$. Create a pipe between t and t^-, and subdivide the pipe t^-t with two new clusters t^-_{leaf} and t_{leaf} in $T[v^-]$ and $T[v]$, respectively. The degree of clusters t or t^- might increase to 4; and so we adjust H so that both trees are binary. The image graph is now split into groups $G_{u^-v^-}$ and G_{uv}.

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 cluster-split operation at u, followed by a sequence of cluster-splits along the chain C_0 from ℓ_0 to u, and subdivision operations that create the new leaf clusters 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 $O(\log n + k)$. Then, we can split G_{uv} in $O(\min(|\mathcal{B}^-|, |\mathcal{B}^+|) + |\mathcal{V}^0| \log n + \log n)$ time.

2.5.4 Analysis of the spur-elimination algorithm

Lemma 2.5.4. *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 clumps, and let $\varphi = 2\sigma + \beta$. Initially, $\varphi = O(m)$ 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 cluster of degree two; the update of supporting data structures takes $O(\log m)$ time. Interior clusters are created only when they contain a spur, so at most $O(m)$ interior clusters 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 clumps of a group. We can now decide whether P' is weakly simple in $O(n)$ time similarly to [CEX15, Section 3.3]. If P' is weakly simple, then the benchmark-to-benchmark paths in each group G_{uv} can be perturbed into disjoint paths, and they traverse the rectangle R_{uv} from D_u to D_v, in a specific order. The Euler tours of the leaves in the clumps $C(u)$ and $C(v)$ determine this ordering uniquely, up to permutations of the paths between the same pair of leaves. Such orderings can be computed in $O(n)$ time for all groups. By concatenating the benchmark-to-benchmark paths at each cluster according to the these orderings, we obtain a unique 2-regular graph Q', in which the vertices are the benchmarks and the edges represent benchmark-to-benchmark paths. Polygon P' is weakly simple if and only if Q' is connected and visits the benchmarks in the same cyclic order as P'. These properties can be verified by a simple traversal of P' and Q' in $O(n)$ time. This completes the proof of part 1 of Theorem 2.1.1.

2.6 Perturbing weakly simple polygons into simple polygons

In Sections 2.3–2.5, 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 at 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 [CEX15, 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 pipes of H (cf. Section 2.2). In the absence of forks, every edge lies in a pipe, 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 2.3–2.5, 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 clumps.

For $n \geq 3$, let $P = (p_0, \ldots, p_{n-1})$ be a weakly simple polygon with image graph H. Assume that the sober clusters of H are partitioned into a set of disjoint clumps satisfying invariants (I1)–(I4) such that every bar is either entirely in a clump or outside of all clumps. 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 H has no vertical pipes. In each pipe uv of H outside of clumps, the above-below relationship yields a total ordering over the edges of Q that contain uv. For each bar b outsize of clumps, the total orders of the pipes along b are consistent (since the above-below relationship between two edges is the same in every rectangle). 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 ap-

proximate simple polygon Q' as follows; refer to Figure 2.28. For every bar $b = uv$ of H outside of clumps, 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 clump $C(u)$, denote by $R(u)$ the union of all disks D_v, $v \in C(u)$, and all rectangles between clusters in $C(u)$. If $C(u)$ contains only the cluster u, then $R(u) = D_u$, but $R(u)$ is always simply connected since $C(u)$ induces a tree $T[u]$. For each clump $C(u)$, 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, $0 < \varepsilon < \varepsilon_0$.

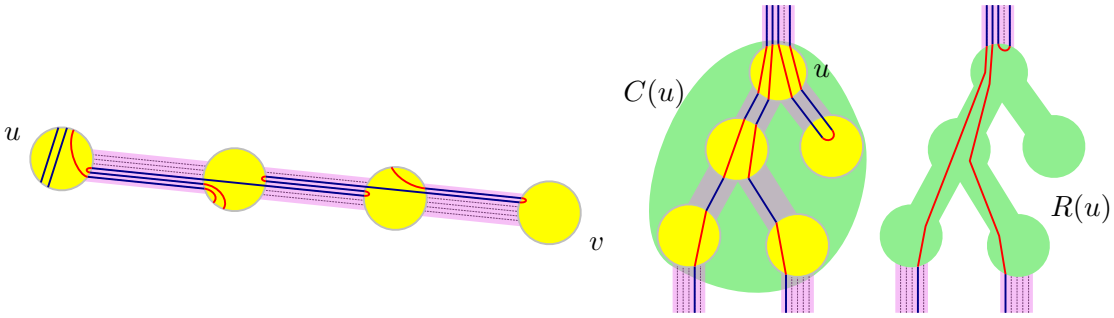


Figure 2.28: Construction of a simple polygon $Q' \in \Psi(P)$ from a bar-signature. Left: Bar uv of a simple polygon obtained from an order compatible with the polygon shown in Figure 2.2(c). Right: maximal paths of Q and Q' inside clumps.

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 clusters. Specifically, P'' is represented by an image graph H'', 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 rectangle R_{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 spur-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 2.5.3), 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 2.5.2), we reverse each of the steps in the following manner. Recall that if a new group $G_{\ell^-\ell^+}$ was created, then every path $[t'_1; t'_2] \in \mathcal{B}'$ was created from two paths $[t_1; u]$ and $[t_2; u]$. Use the ordering of $\mathcal{B}'$ to order the paths that originated $\mathcal{B}'$ 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 2.26). Identify the leftmost path in the pipe 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 cluster 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 2.4.1, 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 2.3.1) 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 pipes between adjacent clumps. 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 pipes that get merged.

Updating the bar-signature when we reverse an L-shortcut operation is a bit more challenging. Determining the edge order in pipes vw and vu_1 can be trivially done by just concatenating the order of merged pipes. 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 2.3.1 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 Δ_b, and when bar-simplification terminates, only the cross-chains remain in the interior of Δ_b. Let K denote the set of cross-chains of Δ_b. The bar-signature yields a linear order (from left to right) of K; and the cross-chains subdivide Δ_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 phase (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 above-below relationship between the edges

of each cross-chain is uniquely determined by Lemma 2.3.2. Since the ordering of each subset guarantees that its paths can be realized without crossing, any linear extension of these ordering produces a bar-signature of a simple polygon.

Preprocessing. The clump 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 $[a, c]$ in a bar b is subdivided into $[a, b, c]$, where $[b, c]$ is in Δ_b, we can assign $[a, c]$ to the same position of $[b, c]$ in the ordering of edges in b. The crimp-reduction operations can be reversed by making the three edges that form a crimp consecutive in the ordering, as in the proof of Lemma 2.3.1. This completes the proof of Theorem 2.1.1.

2.7 Single vertex foldability

Recall (from Chapter 1) that a crease pattern is the plane graph formed by creases in the paper. In a *single-vertex* crease pattern, the paper is an infinite orientable manifold with zero Gaussian curvature at all points except possibly for the origin where the curvature might be not defined. The uncreased regions of paper are infinite wedges that must be realized flat in the folding. We call the paper *flat* if the sum of angles of all wedges is 360°. Creases are rays from the origin in the paper. Each crease can be associated to a *folding angle* which corresponds to the difference between the angles formed by the normals of two adjacent faces bounding the crease in the desired state and in the unfolded state. In Flat-Foldability these angles are either $-180°$ or 180°, but in general they can be any value in between.

In [DO07], the following is stated as Open Problem 12.1: Provide necessary and sufficient conditions for a single vertex crease pattern with assigned folding angles to be realizable as a 3D single-vertex fold, both for flat paper, and for arbitrary incident paper angle. We show that this problem can be solved via a reduction to recognizing weak embeddings.

Corollary 2.7.1. *Given a single-vertex crease pattern with prescribed folding angles, deciding whether it can be realized as a 3D single-vertex fold can be done in* $O(n \log n)$

time.

Proof. We define a folding isometry of a 3D single-vertex fold similarly to Chapter 1. Place the vertex at the origin and fix one of the faces in the xy-plane arbitrarily. The positions of the remaining faces are then defined by continuity and the prescribed folding angle. If the conditions in Theorem 12.2.14 in [DO07] are met (the combined rotations defined by the folding angles result in the identity), then the defined folding isometry is continuous. This can be checked in $O(n)$ time. Take the intersection between the paper and the unit sphere centered at the origin. This defines a polygon on $\mathbb{S}^2$. Note that the intersection of a face and the sphere defines an arc of a great circle. Project the polygon in the plane using gnomonic projection which maps great circles to straight lines. The result is a polygon P that is weakly simple if and only if the 3D single-vertex fold is realizable without crossings. □

2.8 Proof of Theorem 2.1.2

This section generalizes Theorem 2.1.1 to drawings of max-degree-2 graphs in arbitrary surfaces as is done by Chang et al. [CEX15, Section 8]. First we show a reduction to regular degree-2 graphs.

Lemma 2.8.1. *Let $\varphi : G \to H$ be an instance containing a polygonal path p. Let the instance $\varphi' : G' \to H$ be obtained by replacing p by the polygon P constructed by identifying the endpoint of p and p^{-1}. Then, φ and φ' are equivalent.*

Proof. Assume that there exist an embedding that approximates φ. A sufficiently small neighborhood of p must contain no points apart from point in p. Then, we can trivially connect the endpoints of p by a simple path that stays in this neighborhood of p, thus obtaining an embedding approximating φ'. Now assume that there exist an embedding that approximates φ'. The simple polygon that approximates P can be partitioned into two simple paths that approximate p. By simply deleting the image of p^{-1}, we obtain an embedding approximating φ. □

As pointed out in [CEX15], cluster-expansion and old-bar-expansion only rely

on the fact that G is regular degree-2, and, thus, also work for disjoint cycles. The same is true for the new operations apart from the following case. Notice that if a cycle is entirely contained in a bar, then the preprocessing step will replace it by a copy of C_2 whose vertices are mapped to the extremal vertices of the original cycles (resp., leftmost and rightmost points considering a horizontal bar). Let $\mathcal{C}$ be the set of all such cycles. Then, bar-simplification will not report negative cases when an edge in M_{cr} (a longest edge of a cross-chain) is contained in the interior of an edge of a cycle in $\mathcal{C}$. We can check for this case after the preprocessing in $O(n \log n)$ time by building an interval tree storing edges in M_{cr} and testing for overlaps with cycles in $\mathcal{C}$. If such situation does not happen, we can delete all cycles in $\mathcal{C}$. This produces an equivalent instance as follows. Clearly if the original instance is positive, deleting $\mathcal{C}$ will not affect that. Notice that all cross chains can be sorted from left to right and that, since no cycle in $\mathcal{C}$ contain an edge in M_{cr}, we can determine for every cycle in $\mathcal{C}$ the cross chains that must be to the left and to the right of the cycle. If a bar has k cross chains, such cross chains partition the other polygonal paths in the same bar into $k + 1$ intervals depending on which side of the cross chains the path is embedded. If the instance without $\mathcal{C}$ is positive, then there exists an embedding in which, for all interval defined by cross chains, no edge of a top chain is below an edge of a bottom chain. In particular, the embedding described in Section 2.6 has this property by construction. Then, one can always draw the cycles in $\mathcal{C}$ in an appropriate interval below all top chains and above all bottom chains.

Further, the termination of the algorithm has to be changed to handle regular degree-2 graphs. Instead of a single cycle, the algorithm may end up with disjoint and/or nested simple cycles. It is possible to check if such components can be approximated by an embedding in $\mathbb{R}^2$ using observations in [CEX15]. Each disjoint set of cycles can be treated separately. Each set of nesting cycles contain identical cycles. The existence of an approximating embedding depends only on the "winding number", i.e., how many times a cycle c in G wraps around the corresponding simple cycle C in H. Since the strip system of C is homeomorphic to an annulus, report a negative instance if the winding number of each cycle is different than 1. Otherwise,

report that φ is a weak embedding. Apart from these changes, all other details of our algorithms directly generalize. This concludes the proof of Theorem 2.1.2.

Chapter 3

Weak Embeddings

This chapter presents an efficient algorithm for recognizing weak embeddings. A polynomial-time algorithm for recognizing weak embeddings was recently found by Fulek and Kynčl [FK18], which reduces to solving a system of linear equations over $\mathbb{Z}_2$. It runs in $O(m^{4\omega})$ time, where $\omega \in [2, 2.373)$ is the matrix multiplication exponent and m is the number of edges of G. We improve the running time to $O(nm \log n)$, where n is the number of vertices of G. Our algorithm is also conceptually simpler than [FK18]: We perform a sequence of *local operations* that gradually "untangles" the image $\varphi(G)$ into an embedding $\psi(G)$, or reports that φ is not a weak embedding. It generalizes a recent technique developed for the case that G is a cycle and the embedding is a simple polygon [AAET17], and combines local constraints on the orientation of subgraphs directly, thereby eliminating the need for solving large systems of linear equations. The results in this chapter are joint work with Radoslav Fulek and Csaba D. Tóth published in [AFT18].

3.1 Introduction

We recall some definitions from Chapter 1. An *embedding* $\psi : G \to M$ is a continuous piecewise linear injective map where the graph G is considered as a 1-dimensional simplicial complex. Equivalently, an embedding maps the vertices into distinct points and the edges into interior-disjoint Jordan arcs between the corresponding

vertices. We would like to decide whether a given map $\varphi : G \to M$ can be "perturbed" into an embedding $\psi : G \to M$. Let M be a 2-dimensional manifold equipped with a metric. A continuous piecewise linear map $\varphi : G \to M$ is a *weak embedding* if, for every $\varepsilon > 0$, there is an embedding $\psi_\varepsilon : G \to M$ with $\|\varphi - \psi_\varepsilon\| < \varepsilon$, where $\|.\|$ is the uniform norm (i.e., sup norm).

In some cases, it is easy to tell whether $\varphi : G \to M$ is a weak embedding: Every embedding is a weak embedding; and if φ maps two edges into Jordan arcs that cross transversely, then φ is not a weak embedding. The problem becomes challenging when φ maps several vertices (edges) to the same point (Jordan arc), although no two edges cross transversely. This scenario arises in applications in clustering, cartography, and visualization, where nearby vertices and edges are often bundled to the same point or overlapping arcs, due to data compression, graph semantics, or low resolution. A *cluster* in this context is a subgraph of G mapped by φ to the single point in M.

The recognition of weak embeddings turns out to be a purely combinatorial problem independent of the global topology of the manifold M (as noted in [CEX15]). The key observation here is that we are looking for an embeddding in a small neighborhood of the image $\varphi(G)$, which can be considered as the embedding of some graph H. As such, we can replace M with a neighborhood of an embedded graph in the formulation of the problem.

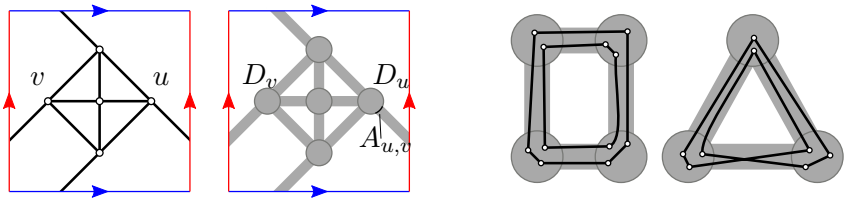


Figure 3.1: (a) An embedding of $H = K_5$ in the torus. (b) Strip system $\mathcal{H}$ of the embedding of H. (c) A weak embedding where G is disconnected and $H = C_4$. (d) A negative instance where $G = C_8$ and $H = C_3$.

Problem Statement and Results. In Sections 3.2–3.5, we make two simplifying assumptions: We assume that M is an *orientable* 2-manifold, and $\varphi : G \to H$

is a *simplicial map* for some graph H embedded in M. A map $\varphi : G \to H$ is *simplicial* if it maps the vertices of G to vertices of H and the edges of G to edges or vertices of H such that incidence relations are preserved (in particular, the image of an edge cannot pass through the image of a vertex). We explain below how to drop the assumption that $\varphi : G \to H$ is simplicial at the expense of increasing the running time of our algorithms. We extend our results to nonorientable surfaces in Section 3.6.

An *embedded graph* H in an orientable 2-manifold M is an abstract graph together with a *rotation system* that specifies, for each vertex of H, the ccw cyclic order of incident edges. The *strip system* $\mathcal{H}$ of H (a.k.a. the *thickening* of H) is a 2-manifold with boundary constructed as follows (Fig. 3.1(a)–(b)): For every $u \in V(H)$, create a topological disk D_u, and for every edge $uv \in E(H)$, create a rectangle R_{uv}. For every D_u and R_{uv}, fix an arbitrary orientation of ∂D_u and ∂R_{uv}, respectively. Partition the boundary of ∂D_u into $\deg(u)$ arcs, and label them by $A_{u,v}$, for all $uv \in E(H)$, in the cyclic order around ∂D_u determined by the rotation of u in the embedding of H. Finally, the manifold $\mathcal{H}$ is obtained by identifying two opposite sides of every rectangle R_{uv} with $A_{u,v}$ and $A_{v,u}$ via an orientation preserving homeomorphism (i.e., consistently with the chosen orientations of $\partial R_{uv}, \partial D_u$ and ∂D_v). See Fig. 3.1(a)–(b). If $\varphi : G \to M$ is continuous piecewise linear map, then $\varphi(G)$ is the embedding of a some graph H in M, and an ε-neighborhood of $\varphi(G)$ in M is homeomorphic to $\mathcal{H}$ for a sufficiently small $\varepsilon > 0$. Consequently, $\varphi : G \to M$ is a weak embedding if and only if $\varphi : G \to \mathcal{H}$ is a weak embedding. We may further assume that φ_ε maps every edge uv to a Jordan arc that crosses the boundaries of D_u (resp., D_v) precisely once (cf. [CEX15, Lemma B2] and [FK18, Section 4]).

We can now formulate an instance of the weak embeddability problem as a simplicial map $\varphi : G \to H$ (for short, φ), where G is an abstract graph and H is an embedded graph. The simplicial map $\varphi : G \to H$ is a *weak embedding* if there exists an embedding $\psi_\varphi : G \to \mathcal{H}$ that maps each vertex $v \in V$ to a point in $D_{\varphi(v)}$, and each edge $uv \in E(G)$ to a Jordan arc in $D_{\varphi(u)} \cup R_{\varphi(u)\varphi(v)} \cup D_{\varphi(v)}$ that has a connected intersection with each of $D_{\varphi(u)}$, $R_{\varphi(u)\varphi(v)}$, and $D_{\varphi(v)}$, and $R_{\varphi(u)\varphi(v)} = \emptyset$

if $u = v$. We say that the embedding ψ_φ *approximates* φ. Our main results is the following.

Theorem 3.1.1.

(i) *Given an abstract graph G with m edges, an embedded graph H, and a simplicial map $\varphi : G \to H$, we can decide in $O(m \log m)$ time whether φ is a weak embedding.*

(ii) *If $\varphi : G \to H$ is a weak embedding, then for every $\varepsilon > 0$ we can also find an embedding $\psi_\varepsilon : G \to M$ with $\|\varphi - \psi_\varepsilon\| < \varepsilon$ in $O(m \log m)$ time.*

Throughout this chapter we assume that G has n vertices and m edges. In the plane (i.e., $M = \mathbb{R}^2$), only planar graphs admit weak embeddings hence $m = O(n)$, but our techniques work for 2-manifolds of arbitrary genus, and G may be a dense graph. Our result improves the running time of the previous algorithm [FK18] from $O(m^{2\omega}) \leq O(m^{4.75})$ to $O(m \log m)$, where $\omega \in [2, 2.373)$ is the matrix multiplication exponent. It also improves the running times of several recent polynomial-time algorithms in special cases, e.g., when the embedding of G is restricted to a given isotopy class [Ful17], and H is a path [ADLDBF17] (see below).

Extension to Nonsimplicial Maps and Nonorientable Manifolds. If $\varphi : G \to H$ is a continuous map (not necessarily simplicial) that is injective on the edges (each edge is a Jordan arc), we may assume that $\varphi(V(G)) \subseteq V(H)$ by subdividing the edges in $E(H)$ with at most $n = |V(G)|$ new vertices in $V(H)$ if necessary. Then φ maps every edge $e \in G$ to a path of length $O(n)$ in H. By subdividing the edges $e \in E(G)$ at all vertices in $V(H)$ along $\varphi(e)$, we reduce the recognition problem to the regime of simplicial maps (Theorem 3.1.1). The total number of vertices of G may increase to $O(mn)$ and the running time to $O(mn \log(mn))$.

Corollary 3.1.2. *Given an abstract graph G with m edges, an embedded graph H with n vertices, and a piecewise linear continuous map $\varphi : G \to H$ that is injective on the interior of every edge in $E(G)$, we can decide in $O(mn \log(mn))$ time whether φ is a weak embedding.*

For example, this applies to straight-line drawings in $\mathbb{R}^2$ if the edges may pass through vertices.

Corollary 3.1.3. *Given an abstract graph G with n vertices and a map $\varphi : G \to \mathbb{R}^2$ where every edge is mapped to a straight-line segment, we can decide in $O(n^2 \log n)$ time whether φ is a weak embedding.*

In Section 3.6, we extend Theorem 3.1.1 and Corollary 3.1.2 to *nonorientable* surfaces with minor changes in the combinatorial representations, using a signature $\lambda : E(H) \to \{-1, 1\}$ to indicate whether an edge uv (and R_{uv}) is orientation-preserving or -reversing [BD09].

Related Previous Work. The study of weak embeddings lies at the interface of several independent lines of research in mathematics and computer science. In topology, the study of weak embeddings and its higher dimensional analogs were initiated by Sieklucki [Sie69] in the 1960s. One of his main results [Sie69, Theorem 2.1] implies the following. Given a graph G and an embedded path H, *every* simplicial map $\varphi : G \to H$ is a weak embedding if and only if every connected component of G is a subcubic graph with at most one vertex of degree three. It is easy to see that an analogous statement is false when G has maximum degree 2 (i.e., G is a union of vertex disjoint paths and cycles). Chapter 2 describes related work and results for such case.

Finding efficient algorithms for the recognition of weak embeddings $\varphi : G \to H$, where G is an arbitrary graph, was posed as an open problem in [AAET17, CEX15, CDBPP09]. The first polynomial-time solution for the general version follows from a recent variant [FK18] of the Hanani-Tutte theorem [Han34, Tut70], which was conjectured by M. Skopenkov [Sko03] in 2003 and in a slightly weaker form already by Repovš and A. Skopenkov [RS98] in 1998. However, this algorithm reduces the problem to a system of $O(m)$ linear equations over $\mathbb{Z}_2$, where $m = |E(G)|$. The running time for simplicial maps is dominated by solving this system in $O(m^{2\omega}) \leq O(m^{4.75})$ time, where $\omega \in [2, 2.373)$ is the matrix multiplication exponent; cf. [Wil12].

Weak embeddings of graphs also generalize various graph visualization models such as the recently introduced *strip planarity* [ADLDBF17] and *level planarity* [JLM98]; and can be seen as a special case [AL16] of the notoriously difficult *cluster-planarity* (for short, *c-planarity*) [FCE95a, FCE95b], whose tractability remains elusive despite many attempts by leading researchers.

Outline. Our results rely on ideas from [ADLDBF17, CEX15, CDBPP09] and [FK18]. Recall that, to distinguish the graphs G and H, we use the convention that G has *vertices* and *edges*, and H has *clusters* and *pipes*. A cluster $u \in V(H)$ corresponds to a subgraph $\varphi^{-1}[u]$ of G, and a pipe $uv \in E(H)$ corresponds to a set of edges $\varphi^{-1}[uv] \subseteq E(G)$.

The main tool in our algorithm is a local operation, called "cluster expansion," which generalizes similar operations introduced previously for the case that G is a cycle. Given an instance $\varphi : G \to H$ and a cluster $u \in V(H)$, it modifies u and its neighborhood (by replacing u with several new clusters and pipes) such that weak embeddability is invariant under the operation in the sense that the resulting new instance $\varphi' : G' \to H'$ is a weak embedding if and only if $\varphi : G \to H$ is a weak embedding. Our operation increases the number of clusters and pipes, but it decreases the number of "ambiguous" edges (i.e., multiple edges in the same pipe). The proof of termination and the running time analysis use potential functions.

In a preprocessing phase, we perform a cluster expansion operation at each cluster $u \in V(H)$. The main loop of the algorithm applies another operation, "pipe expansion," for two adjacent clusters $u, v \in V(H)$ under certain conditions. It merges the clusters u and v, and the pipe $uv \in E(H)$ between them, and then invokes cluster expansion. If any of these operations finds a local configuration incompatible with an embedding, then the algorithm halts and reports that φ is not a weak embedding (this always corresponds to some nonplanar subconfiguration since the neighborhood of a single cluster or pipe is homeomorphic to a disk). We show that after $O(m)$ successive operations, we obtain an irreducible instance for which our problem is easily solvable in $O(m)$ time. Ideally, we end up with $G = H$

(one vertex per cluster and one edge per pipe), and $\varphi = \mathrm{id}$ is clearly an embedding. Alternatively, G and H may each be a cycle (possibly G winds around H multiple times), and we can decide whether φ is a weak embedding in $O(m)$ time by a simple traversal of G. If G is disconnected, then each component falls into one of the above two cases (Fig. 3.1(c)–(d)), i.e., the case when $\varphi = \mathrm{id}$ or the case when $\varphi \neq \mathrm{id}$.

The main challenge was to generalize previous local operations (that worked well for cycles [CEX15, CDBPP09, Sko03]) to arbitrary graphs. Our expansion operation for a cluster $u \in V(H)$ simplifies each component of the subgraph $\varphi^{-1}[u]$ of G independently. Each component is planar (otherwise it cannot be perturbed into an embedding in a disk D_u). However, a planar (sub)graph with k vertices may have $2^{O(k)}$ combinatorially different embeddings: some of these may or may not be compatible with adjacent clusters. The embedding of a (simplified) component C of $\varphi^{-1}[u]$ depends, among other things, on the edges that connect C to adjacent clusters. The *pipe-degree* of C is the number of pipes that contain its incident edges. If the pipe-degree of C is 3 or higher, then the rotation system of H constrains the embedding of C. If the pipe-degree is 2, however, then the embedding of C can only be determined up to a reflection, unless C is connected by *two* independent edges to a component in $\varphi^{-1}[v]$ whose orientation is already fixed; see Fig. 3.2.

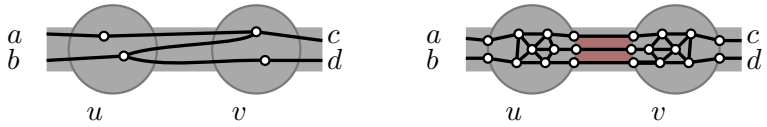


Figure 3.2: Two adjacent clusters, u and v, that each contain two components of pipe-degree 2 (left). These components merge into a single component in $D_u \cup R_{uv} \cup D_v$. In every embedding, the order of the pipe-edges a, b determines the order of the pipe-edges c, d. The operation $\mathsf{pipeExpansion}(uv)$ transforms the component on the left to two wheels connected by three edges (a so-called thick edge) shown on the right.

We need to maintain the feasible embeddings of the components in all clusters efficiently. In [FK18], this problem was resolved by introducing 0-1 variables for the components, and aggregating the constraints into a system of linear equations over $\mathbb{Z}_2$, which was eventually resolved in $O(m^{2\omega}) \leq O(m^{4.75})$ time. We improve the

running time to $O(m \log m)$ by maintaining the feasible embeddings simultaneously with our local operations.

Another challenge comes from the simplest components in a cluster $\varphi^{-1}[u]$. Long chains of degree-2 vertices, with one vertex per cluster, are resilient to our local operations. Their length may decrease by only one (and cycles are irreducible). We need additional data structures to handle these "slowly-evolving" components efficiently. We use a dynamic heavy-path decomposition data structure and a suitable potential function to bound the time spent on such components.

Organization. In Section 3.2, we introduce additional terminology for an instance $\varphi : G \to H$, and show how to modify the subgraphs of G within each cluster to reduce the problem to an instance in "normal form," a "simplified form," and introduce a combinatorial representation a weak embedding that we use in our algorithm. The simplification step relies on the concept of SPQR-trees, developed in [DT96] for the efficient representation of combinatorial embeddings of a graph, which we also review in this section.

In Section 3.3, we present the cluster expansion and pipe expansion operations and prove that weak embeddibility is invariant under both operations. We use these operations repeatedly in Section 3.4 to decide whether a simplicial map $\varphi : G \to H$ is a weak embedding. Section 3.5 discusses how to reverse a sequence of operations to perturb a weak embedding into an embedding. The adaptation of our results to nonorientable surfaces is discussed in Section 3.6.

3.2 Preliminaries

In this section, we describe modifications within the clusters of a simplicial map $\varphi : G \to H$ to bring it to "normal form" (properties (P1)–(P2)) and "simplified form" (properties (P3)–(P4)). These properties allow for a purely combinatorial representation of weak embeddability (in terms of permutations), which we use in the proof of correctness of the algorithms in Sections 3.3 and 3.4.

Definitions. Two instances $\varphi : G \to H$ and $\varphi' : G' \to H'$ are called *equivalent* if φ is a weak embedding if and only if φ' is a weak embedding. We call an edge $e \in E(G)$ a *pipe-edge* if $\varphi(e) \in E(H)$, or a *cluster-edge* if $\varphi(e) \in V(H)$. For every cluster $u \in V(H)$, let G_u be the subgraph of G induced by $\varphi^{-1}[u]$. For every pipe $uv \in E(H)$, $\varphi^{-1}[uv]$ stands for the set of pipe-edges mapped to uv by φ.

The *pipe-degree* of a connected component C of G_u, denoted pipe-deg(C), is the number of pipes that contain some edge of G incident to C. A vertex v of G_u is called a *terminal* if it is incident to a pipe-edge. For an integer $k \geq 3$, the k-vertex *wheel* graph W_k is a join of a *center* vertex c and a cycle of $k-1$ *external* vertices. Refer to [Die17] for standard graph theoretic terminology (e.g., cut vertex, 2-cuts, biconnectivity).

3.2.1 Normal Form

An instance $\varphi : G \to H$ is in *normal form* if every cluster $u \in V(H)$ satisfies:

(P1) Every terminal in G_u is incident to exactly one cluster-edge and one pipe-edge.

(P2) There are no degree-2 vertices in G_u.

We now describe subroutine $\mathsf{normalize}(u)$ that, for a given instance $\varphi : G \to H$ and a cluster $u \in V(H)$, returns an equivalent instance $\varphi' : G' \to H$ such that u satisfies (P1)–(P2); refer to Fig. 3.3(a)–(b).

$\mathsf{normalize}(u)$. Input: an instance $\varphi : G \to H$ and a cluster $u \in V(H)$.

Subdivide every pipe-edge pq where $\varphi(p) = u$ into a path (p, p', q) such that $\varphi'(p) = \varphi'(p') = \varphi'(q) = u$. Note that the new vertex p' is a terminal in G and a leaf in G_u (i.e., $\deg_{G_u}(p') = 1$). Successively suppress every vertex p of G_u with $\deg_{G_u}(p) = 2$ by merging its incident edges. If this creates a loop, delete the loop.

Lemma 3.2.1. *Given an instance $\varphi : G \to H$ and a cluster $u \in V(H)$, the instance $\varphi' = \mathsf{normalize}(u)$ and φ are equivalent and u satisfies (P1)–(P2) in φ'. The subroutine runs in $O(\sum_{p \in V(G_u)} \deg_G(p))$ time. By successively applying $\mathsf{normalize}$ to*

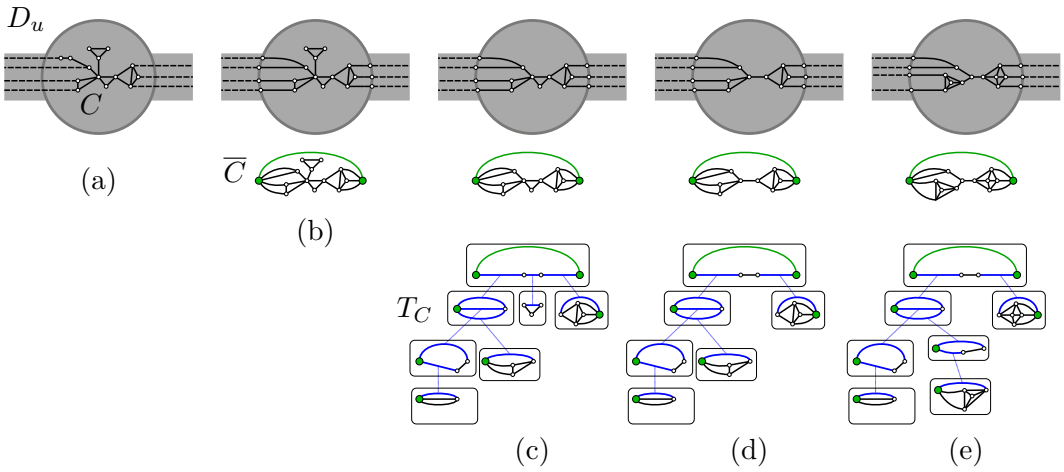


Figure 3.3: Changes in a cluster caused by normalize and simplify. (a) Input, (b) after normalize, (c) after first part of step 1, (d) after step 1, and (e) after step 2 of subroutine simplify. Dashed lines, green dots, green lines, and blue lines represent pipe-edges, pipe-vertices, edges in $\overline{E}_C$, and virtual edges, respectively.

all clusters in $V(H)$, we obtain an equivalent instance in normal form in $O(|E(G)|)$ time.

Proof. The instances φ and φ' are clearly equivalent since (i) we can always replace the embedding of an edge by a path and vice-versa, and (ii) a loop can always be deleted or added to a vertex in an embedding. There are $O(\sum_{p\in V(G_u)} \deg_G(p))$ pipe-edges incident to a vertex in u. Hence the subroutine performs $O(\sum_{p\in V(G_u)} \deg_G(p))$ subdivisions. There are at most $|E(G_u)| = O(\sum_{p\in V(G_u)} \deg_G(p))$ degree-2 vertices in G_u. By construction, the resulting graph G'_u satisfies (P1)–(P2). All changes are local and applying normalize to u does not change properties (P1)–(P2) in other clusters. Therefore, we can obtain the normal form of φ in $O(|E(G)|)$ time. □

For every cluster $u \in V(H)$, the graph G_u may have several components. For each component C, we define a multigraph $\overline{C}$ that represents the interactions of C with vertices in adjacent clusters. Specifically, for each component C of G_u of a cluster $u \in V(H)$ satisfying (P1)–(P2), we define the multigraph $\overline{C}$ in two steps as follows; refer to Fig. 3.3(b).

1. For every pipe $uv \in E(H)$ incident to u, create a new vertex v', called *pipe-vertex*, and identify all terminal vertices of C incident to some edge in $\varphi^{-1}[uv]$

with v' (this may create multiple edges incident to v').

2. If pipe-deg$(C) = 2$, connect the two pipe-vertices with an edge e and let $\overline{E}_C = \{e\}$. If pipe-deg$(C) \geq 3$, connect all pipe-vertices by a cycle in the order determined by the rotation of u and let $\overline{E}_C$ be the set of edges of this cycle.

Let $\overline{C} = (V(\overline{C}), E(\overline{C}))$, where $V(\overline{C})$ consists of nonterminal vertices in $V(C)$ and pipe-deg(C) pipe-vertices, and $E(\overline{C})$ consists of the edges induced by nonterminal vertices in $V(C)$, (multi) edges created in step 1 (each of which corresponds to an edge in $E(C)$) and edges in $\overline{E}_C$. It is clear that every embedding of $\overline{C}$ can be converted into an embedding of C such that the rotation of the pipe vertices in $\overline{C}$ determines the cyclic order of terminals along the facial walk of the outer face of C.

3.2.2 SPQR-Trees

SPQR-trees were introduced by Di Battista and Tamassia [DT96] for an efficient representation of all combinatorial plane embeddings of a graph. Let G be a biconnected planar graph. The SPQR-tree T_G of G represents a recursive decomposition of G defined by its (vertex) 2-cuts. A deletion of a 2-cut $\{u, v\}$ disconnects G into two or more components $C_1, \ldots, C_i$, $i \geq 2$. A *split component* of $\{u, v\}$ is either an edge uv or the subgraphs of G induced by $V(C_j) \cup \{u, v\}$ for $j = 1, \ldots, i$. The tree T_G captures the recursive decomposition of G into split components defined by 2-cuts of G. A node μ of T_G is associated with a multigraph called skeleton(μ) on a subset of $V(G)$, and has a *type* in {S,P,R}. If the type of μ is S then skeleton(μ) is a cycle of 3 or more vertices. If the type of μ is P then skeleton(μ) consists of 3 or more parallel edges between a pair of vertices. If the type of μ is R then skeleton(μ) is a 3-connected graph on 4 or more vertices. An edge in skeleton(μ) is *real* if it is an edge in G, or *virtual* otherwise. A virtual edge connects the two vertices of a 2-cut, u and v, and represents a subgraph of G obtained in the recursive decomposition, containing a uv-path in G that does not contain any edge in skeleton(μ). Two nodes μ_1 and μ_2 of T_G are adjacent if skeleton(μ_1) and skeleton(μ_2) share exactly two ver-

tices, u and v, that form a 2-cut in G. Each virtual edge in skeleton(μ) corresponds to a pair of adjacent nodes in T_G. No two S nodes (resp., no two P nodes) are adjacent. Every edge in $E(G)$ appears in the skeleton of exactly one node. The tree T_G has $O(|E(G)|)$ nodes and it can be computed in $O(|E(G)|)$ time [DT96].

It is also known that T_G represents all combinatorial embeddings of G in $\mathbb{R}^2$ in the following manner [DT96]. Choose a root node for T_G and an embedding of its skeleton. Then successively replace each virtual edge uv by the skeleton of the corresponding node μ minus the virtual edge uv in skeleton(μ). In each step of the recursion, if μ is of type R, skeleton(μ) can be flipped (reflected) around u and v, and if μ is of type P, the parallel edges between u and v can be permuted arbitrarily.

3.2.3 Combinatorial Representation of Weak Embeddings

Given an embedding $\psi_\varphi : G \to \mathcal{H}$, where φ is in normal form, we define a combinatorial representation π_φ as the set of total orders of edges in R_{uv}, for all pipes $uv \in E(H)$. Specifically, for every pipe $uv \in E(H)$, fix an orientation of the boundary of $R(uv)$ (e.g., the one used in the construction of the strip system $\mathcal{H}$). Record the order in which the pipe-edges in the embedding ψ_φ intersect a fixed side of ∂R_{uv} (say, the side $\partial R_{uv} \cap \partial D_u$) when we traverse it in the given orientation. Let $\Pi(\varphi)$ be the set of combinatorial representations π_φ of all embeddings $\psi_\varphi : G \to \mathcal{H}$. Note that $\varphi : G \to H$ is a weak embedding if and only if $\Pi(\varphi) \neq \emptyset$.

Conversely, let π_φ be a set of total orders of edges in R_{uv}, for all pipes $uv \in E(H)$. We show (in Lemma 3.2.2 below) how to use the normal form and SPQR-trees to decide whether $\pi_\varphi \notin \Pi(\varphi)$. We say that two components C_1 and C_2 of G_u **cross** with respect to π_φ if and only if their terminals interleave in the cyclic order around ∂D_u (i.e., there exists no cut in the cyclic order in which all terminals of C_1 appear before all terminals of C_2). If $\pi_\varphi \in \Pi(\varphi)$, then π_φ cannot induce two crossing components in G_u, for any $u \in V(H)$.

Lemma 3.2.2. *Given a set of total orders π_φ, we can decide whether $\pi_\varphi \notin \Pi(\varphi)$ in $O(m)$ time. If $\pi_\varphi \in \Pi(\varphi)$, then we can also find an embedding $\psi_\varphi : G \to \mathcal{H}$ in*

$O(m)$ time.

Proof. Let $\mathcal{H}$ be the strip system for $\varphi : G \to H$. For each pipe $uv \in E(H)$, draw $|\varphi^{-1}[uv]|$ parallel Jordan arcs in R_{uv} connecting ∂D_u and ∂D_v. For each cluster $u \in V(H)$, π_φ defines a ccw cyclic order of terminals around ∂D_u.

Create a graph $\widetilde{C}$ by the union of C and a wheel W_{t_C+1} whose center is a new vertex, and whose external vertices are the terminals of C in the order defined by π_φ. An embedding of C in which the terminals appear in the outer face in the same cyclic order as the one defined by π_φ exists if and only if $\widetilde{C}$ is planar. If $\widetilde{C}$ is not planar, report that $\pi_\varphi \notin \Pi(\varphi)$. Else, embed C inside D_u given the position of the already embedded terminals on ∂D_u. This subdivides D_u into faces. If a component connects terminals in different faces, then two components in G_u cross, hence $\pi_\varphi \notin \Pi(\varphi)$. If no two components cross, we can incrementally embed the components C, as there is always a face of D_u that contains all terminals of C. □

3.2.4 Simplified Form

Given an instance $\varphi : G \to H$ in normal form, we simplify a graph G by removing parts of the graph G_u, for all $u \in V(H)$, that are locally "irrelevant" for the embedding, such as 0-, 1-, and 2-connected components that are not adjacent to edges in any pipe incident to u. Formally, for each component C of G_u, we call a split component defined by a 2-cut $\{p, q\}$ of $\overline{C}$ *irrelevant* if it contains no pipe-vertices. An instance is in *simplified form* if it is in normal form an every $u \in V(H)$ satisfies properties (P3)–(P4) below.

(P3) For every component C of G_u, $\overline{C}$ is biconnected and every 2-cut of $\overline{C}$ contains at least one pipe-vertex.

Assuming that a cluster u satisfies (P3), we define T_C as the SPQR tree of $\overline{C}$ where C is a component of G_u. Given a node μ of T_C, let the *core* of μ, denoted $\mathrm{core}(\mu)$, be the subgraph obtained from $\mathrm{skeleton}(\mu)$ by deleting all pipe-vertices. Property (P4) below will allow us to bound the number of vertices of G_u in terms of its number of terminals (cf. Lemma 3.2.4).

(P4) For every component C of G_u, and every R node μ of T_C, core(μ) is isomorphic to a wheel W_k, for some $k \geq 4$, whose external vertices have degree 4 in G_u.

We now describe subroutine simplify(u) that, for a given instance $\varphi : G \to H$ in normal form and a cluster $u \in V(H)$, returns an instance $\varphi' : G' \to H$ such that u satisfies (P1)–(P4). We break the subroutine into two steps.

simplify(u)**.** Input: an instance $\varphi : G \to H$ in normal form and a cluster $u \in V(H)$. For every component C of G_u, do the following.

(1) If C is not planar, report that φ is not a weak embedding and halt. If pipe-deg(C) = 0, then delete C. Else compute $\overline{C}$, and find the maximal biconnected component $\widehat{C}$ of $\overline{C}$ that contains all pipe-vertices. The component $\widehat{C}$ trivially exists if pipe-deg(C) $\in \{1, 2\}$, and if pipe-deg(C) ≥ 3, it exists since $\overline{E}_C$ forms a cycle containing all pipe-vertices. Modify C by deleting all vertices of $\overline{C} \setminus \widehat{C}$, and update $\overline{C}$ (by deleting the same vertices from $\overline{C}$, as well); refer to Fig. 3.3(b)–(c). Consequently, we may assume that $\overline{C}$ is biconnected and contains all pipe-vertices. Compute the SPQR tree T_C for $\overline{C}$. Set a node μ_r in T_C whose skeleton contain a pipe-vertex as the root of T_C. Traverse T_C using DFS. If a node μ is found such that skeleton(μ) contains no pipe-vertex, let $\{p, q\}$ be the 2-cut of $\overline{C}$ shared by skeleton(μ) and skeleton(parent(μ)). Replace all irrelevant split components defined by $\{p, q\}$ by a single edge pq in C. If p or q now have degree 2, suppress p or q, respectively. Update $\overline{C}$ accordingly, and update T_C to reflect the changes in $\overline{C}$ by changing pq from virtual to real and possibly suppressing p and/or q in skeleton(parent(μ)), which also deletes node μ and its descendants since their skeletons contain edges in the deleted irrelevant split components; refer to Fig. 3.3(c)–(d). Continue the DFS ignoring deleted nodes.

(2) While there is an R node μ in T_C, of a component C in G_u, that does not satisfy (P4), do the following. Let Y be the set of edges in skeleton(μ) adjacent to core(μ) (i.e., edges between a vertex in core(μ) and a pipe-vertex). Since μ is an R node, it represents a 3-connected planar graph, which has a unique combinatorial

embedding (up to reflection). If we contract core(μ) to a single vertex, the rotation of such vertex defines a cyclic order on Y. Let $e_i = pq$, for $p \in V(Y)$, be the i-th edge in this order. Recall that e_i is an edge in skeleton(μ) and therefore represents a subgraph $\overline{C}_{e_i}$ of $\overline{C}$. If e_i is real, $\overline{C}_{e_i}$ is a single edge. Otherwise, $\overline{C}_{e_i}$ contains all split components defined by the 2-cut $\{p, q\}$ that do not contain core(μ) as a subgraph. Do the following changes in C, which will incur changes in $\overline{C}$, as well. Replace core(μ) by the wheel graph $W_{|Y|+1}$, and let r_i be the i-th external vertex in the cycle starting at an arbitrary vertex. For each $\overline{C}_{e_i}$, replace p by a new vertex p_i and add the edge $p_i r_i$. If the resulting p_i has degree 2 in C, suppress p_i. Update T_C to reflect changes in $\overline{C}$ by updating skeleton(μ) and adding an S node between μ and an adjacent node μ_i whose skeleton contained e_i for each virtual $e_i \in Y$ if p_i was not suppressed; refer to Fig. 3.3(d)–(e).

Lemma 3.2.3. *Given an instance $\varphi : G \to H$ in normal form and a cluster $u \in V(H)$, the instance $\varphi' =$ simplify(u) and φ are equivalent and u satisfies (P1)–(P4) in φ'. The operation runs in $O(|E(G_u)|)$ time. By successively applying* simplify *to all clusters in $V(H)$, we obtain an equivalent instance in simplified form in $O(m)$ time.*

Proof. First we prove that u satisfies (P1)–(P4) in φ'. Since φ is in normal form, u satisfies (P1)–(P2) in φ. By construction, u still satisfies (P1)–(P2) in φ'. For (P3), note that after step 1, $\overline{C}$ is biconnected and every node μ of T_C contains a pipe-vertex in its skeleton. Step 2 does not change this property. This implies that core(μ) contains only real edges for every node μ. Suppose for contradiction that there is a 2-cut $\{p, q\}$ such that neither p nor q is a pipe-vertex. Then $\{p, q\}$ must be in core(μ) where μ is a S node, or else either p or q would have been deleted for being in $\overline{C} \setminus \widehat{C}$. Then one split component of $\{p, q\}$ is a path of length two or more. But G'_u has no degree-2 vertex by (P2), a contradiction. Hence, u satisfies (P3) in φ'. By definition, after step 2 u satisfies (P4) in φ'.

We now show that the operation takes $O(|E(G_u)|)$ time. In step 1, planarity testing is done in linear time for each component C of G_u. We obtain $\widehat{C}$ by a DFS.

We compute T_C in $O(|E(C)|)$ time [DT96]. Replacing irrelevant split components by one edge can be done in $O(|E(C)|)$ overall time. In step 2, we can obtain a list of R nodes in $O(|E(C)|)$ time. The changes in step 2 are local, both in C and T_C, and do not influence whether other R nodes satisfy (P4). Step 2 takes $O(|E(G_u)|)$ time overall by processing each R node sequentially. All the changes are local to u and, by successively applying simplify, we obtain a simplified form in $O(|E(G)|)$ time.

Finally, we show that φ and φ' are equivalent. Notice that there is a bijection between the terminals of φ and φ'. We show that $\Pi(\varphi) = \Pi(\varphi')$, i.e., given $\pi_\varphi \in \Pi(\varphi)$, then $\pi_\varphi \in \Pi(\varphi')$ and vice versa. Notice that, given an order π_φ, for each $u \in V(H)$, π_φ induces two components C_1 and C_2 of G_u to cross if and only if the corresponding components C_1' and C_2' of G_u' also cross. Then, it suffices to show that the SPQR trees of $\overline{C}$ and $\overline{C'}$ for corresponding components C of G_u and C' of G_u' represent the same constraints in the cyclic order of terminals around D_u. Step 1 deletes components of pipe-degree 0, which do not pose any restriction on the cyclic order of terminals. The subgraphs represented by irrelevant subtrees in the SPRQ tree of $\overline{C}$ can be flipped independently and, since they do not contain pipe-vertices, their embedding does not interfere with the order of edges adjacent to pipe-vertices. Hence, step 1 does not alter any constraint on the cyclic order of terminals. By construction, step 2 does not change the circular order of edges in core(μ). Replacing core(μ) by a wheel $W_{|Y|+1}$ does not change any of the constraints on the cyclic order of terminals. □

Lemma 3.2.4. *After* simplify(u)*, every component C of G_u contains $O(t_C)$ edges, where t_C is the number of terminals in C.*

Proof. Let us contract every wheel that is a maximal biconnected components in C' by (P4) into a single vertex and remove any loops created by the contraction. Let $\widehat{C}$ denote the resulting component. We have $|E(C)| \leq 5|E(\widehat{C})|$, since the number of contracted edges is at most $4|E(\widehat{C})|$. Indeed, a wheel W_{k+1} has $2k$ edges, which are contracted, and its k external vertices are incident to k edges that are not contracted by (P4). We charge each of these k edges in $E(\widehat{C})$ for two edges of W_k. Then every

edge in $E(\widehat{C})$ receives at most 2 units of charge from each of its endpoints, hence at most 4 units of charge overall. The component $\widehat{C}$ is a tree without degree-2 vertices whose leaves are precisely the terminals of C by (P1)–(P3). Since the number of edges in a tree is at most twice the number its leaves, by the above inequality we have $|E(C)| \leq 5|E(\widehat{C})| \leq 5 \cdot 2 \cdot t_C$, as claimed. □

3.3 Operations

In this section, we present our two main operations, clusterExpansion and pipeExpansion, that we use successively in our recognition algorithm. Given an instance φ and a cluster u in simplified form, operation clusterExpansion(u) either finds a configuration that cannot be embedded locally in the neighborhood of u and reports that φ is not a weak embedding, or replaces cluster u with a group of clusters and pipes (in most cases reducing the number of edges in pipes). It first modifies the embedded graph H, and then handles each component of G_u independently.

Operation pipeExpansion(uv) first merges two adjacent clusters, u and v (and the pipe uv) into a single cluster $\langle uv \rangle$ and invokes clusterExpansion$(\langle uv \rangle)$. We continue with the specifics.

3.3.1 Cluster Expansion

For a cluster $u \in V(H)$ in an instance $\varphi : G \to H$, let the *expansion disk* Δ_u be a topological closed disk containing a single cluster $u \in V(H)$ and intersecting only pipes incident to u.

clusterExpansion(u)**.** Input: an instance $\varphi : G \to H$ in simplified form and a cluster $u \in V(H)$. We either report that φ is not a weak embedding or return an instance $\varphi' : G' \to H'$. The instance φ' is computed incrementally: initially φ' is a copy of φ. Steps 0–3 will insert new clusters and pipes into H' that are within Δ_u, step 4 will determine the rotation system for the new clusters and check whether the rotation system induces any crossing between new pipes within Δ_u, and step 5

brings φ' to its simplified form.

Step 0. For each pipe $uv \in E(H)$ incident to u, subdivide uv by inserting a cluster u_v in H' at the intersection of uv and $\partial\Delta_u$. If $\deg(u) \geq 3$, then add a cycle C_u of pipes through all clusters in $\partial\Delta_u$ (hence the clusters u_v appear along C_u in the order given by the rotation of u); and if $\deg(u) = 2$, then add a pipe between the two clusters in $\partial\Delta_u$. Delete u (and all incident pipes). As a result, the interior of Δ_u contains no clusters.

Step 1: Components of pipe-degree 1. For each component C of G_u such that pipe-deg$(C) = 1$, let uv be the pipe to which the pipe-edges incident to C are mapped to. Move C to the new cluster u_v, i.e., set $\varphi'(C) = u_v$. (For example, see the component in u_x in Fig. 3.4.)

Step 2: Components of pipe-degree 2. For each pair of clusters $\{v, w\}$ adjacent to u, denote by B_{vw} the set of components of G_u of degree 2 adjacent to pipe-edges in $\varphi^{-1}[uv]$ and $\varphi^{-1}[uw]$. For all nonempty sets B_{vw} do the following.

(a) Insert the pipe $u_v u_w$ into H' if it is not already present.

(b) For every component $C \in B_{vw}$, do the following:

(b1) Compute $\overline{C}$ (by (P3), $\overline{C}$ is biconnected). Compute the SPQR tree T_C of $\overline{C}$. Set a node μ_r as the root of T_C so that skeleton(μ_r) contains both pipe-vertices, which we denote by v' and w' (i.e., consistently with Section 3.2). Note that μ_r cannot be of type P, otherwise C would not be connected.

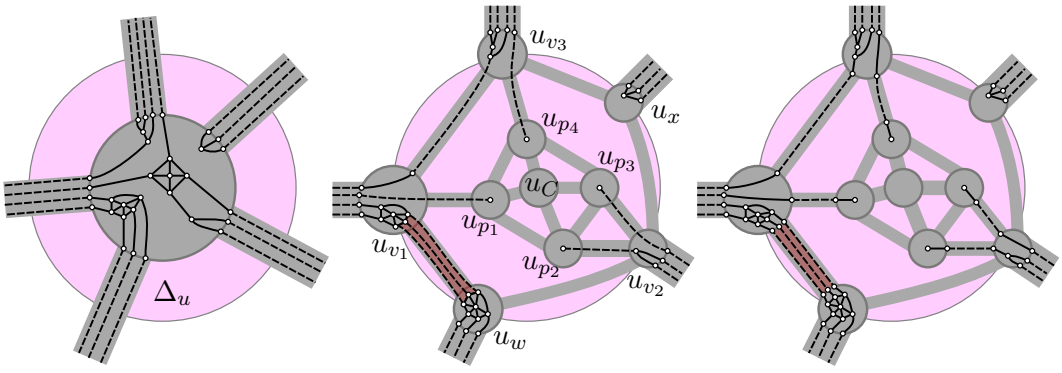


Figure 3.4: Changes in a cluster caused by clusterExpansion. Right: input. Middle: the instance after step 4. Right: output. Red rectangles indicate triples of edges defining a thick edge.

(b2) If μ_r is of type S, then $E(\text{skeleton}(\mu_r)) \setminus \overline{E}_C$ forms a path between the pipe vertices v' and w', that we denote by P, where the first and last edges may be virtual. Notice that path P contains at most 3 edges otherwise (P2) or (P3) would not be satisfied. If $P = (v', w')$ has length 1, then subdivide P into 3 edges $P = (v', p_1, p_2, w')$. If $P = (v', p, w')$ has length 2, then $\{v', p\}$ is a 2-cut in $\overline{C}$ that defines two split components, C_v and C_w, containing v' and w', respectively. Split p into two vertices, p_1 and p_2, connected by an edge so that p_1 (resp., p_2) is adjacent to every vertex in C_v (resp., C_w) that was adjacent to p. The length of P increases to 3, and skeleton(μ_r) changes, as well. Finally, assume that $P = (v', p_1, p_2, w')$ has length 3. Then, in all three cases, the edge p_1p_2 defines an edge cut in C that splits C into two components each incident to a single pipe, one to uv and the other to uw. We define φ' so that it maps each of the two components into u_v or u_w accordingly. (See the components incident to pipe $u_{v_1}u_{v_3}$ in Fig. 3.4.)

(b3) If μ_r is of type R, by (P4), core(μ_r) is a wheel subgraph W_k. Let k_v and k_w, where $k_v + k_w = k - 1$, be the number of edges between W_k and v', and between W_k and w', respectively. Replace W_k by two wheel graphs W_{k_v+4} and W_{k_w+4} connected by three edges so that the circular order of the edges around v' and w' is maintained (recall that an R node has a unique embedding). The triple of edges between W_{k_v+4} and W_{k_w+4} is called a *thick edge*. The thick edge defines a 3-edge-cut that splits C into two components, each with a wheel graph. We define φ' so that each of the two components is mapped to its respective vertex u_v or u_w. (See the components incident to pipe $u_{v_1}u_w$ in Fig. 3.4.)

Step 3: Components of pipe-degree 3 or higher. For all the remaining components C (pipe-deg(C) ≥ 3) of G_u, do the following. Assume C is incident to pipe-edges mapped to the pipes $uv_1, uv_2, \ldots, uv_d$.

(a) Compute $\overline{C}$ and its SPQR tree T_C and let v'_i be the pipe-vertex corresponding to terminals adjacent to edges in uv_i for $i = 1, 2, \ldots, d$. Set the node μ_r as the root of T_C such that skeleton(μ_r) contains the cycle $\overline{E}_C$. The type of μ_r is R, otherwise C would be disconnected.

(b) Changes in H'. By (P4), we have that core(μ_r) is a wheel graph W_k,

where $k-1 \geq d$. For $j = 1, 2, \ldots, k-1$, let p_j be the j-th external vertex of W_k and p_C be its central vertex. Create a copy of W_k using clusters and pipes: Create a cluster u_{p_j} that represents each vertex p_j, a cluster u_C that represents vertex p_C, see Fig. 3.4(middle). Insert the copy of W_k in H'. For every $1 \leq i \leq d$ and $1 \leq j \leq k-1$, insert a pipe $u_{p_j}u_{v_i}$ in H' if an edge $p_j v'_i$ is present in $E(\text{skeleton}(\mu_r))$.

(c) Changes in G'. Delete all edges and the central vertex of W_k from G', which splits C into $k-1$ components. Set $\varphi'(p_j) = u_{p_j}$ for $j \in \{1, \ldots, k-1\}$. By (P4), every cluster u_{p_j} is adjacent to a single cluster, say u_{v_i}, outside of W_k. We modify φ' so that it maps the vertices of the component of C containing p_j to u_{v_i} with the exception of p_j, which is mapped to u_{p_j}. (Note that the cluster u_C and all incident pipes are empty.)

Step 4: Local Planarity Test. Let H_u be the subgraph induced by the newly created clusters and pipes. Let $\widetilde{H_u}$ denote the graph obtained as the union of H_u and a star whose center is a new vertex (not in $V(H_u)$), and whose leaves are the clusters in $\partial\Delta_u$. Use a planarity testing algorithm to test whether $\widetilde{H_u}$ is planar. If $\widetilde{H_u}$ is not planar, report that φ is not a weak embedding and halt. Otherwise, find an embedding of $\widetilde{H_u}$ in which the center of the star is in the outer face. This defines a rotation system for H_u. The rotation system of H' outside of Δ_u is inherited from H.

Step 5: Normalize. Finally, apply normalize to each new cluster in H'. (This step subdivides edges so that (P1) is satisfied as shown in Fig. 3.4(right).)

Lemma 3.3.1. *Given an instance $\varphi : G \to H$ in simplified form containing a cluster u, clusterExpansion(u) either reports that φ is not a weak embedding or produces an instance $\varphi' : G' \to H'$ in simplified form that is equivalent to φ in $O(|E(G_u)| + \deg(u))$ time.*

Proof. Step 5 of the operation receives as input an instance $\varphi^* : G^* \to H^*$ and return an equivalent instance φ' by Lemma 3.2.1. By construction, the new clusters satisfy (P1)–(P4) in φ'. It remains to show that φ^* and φ are equivalent, and to analyze the running time.

First assume that φ is a weak embedding, and so there is an embedding $\psi_\varphi : G \to \mathcal{H}$. We need to show that there exists an embedding $\psi_{\varphi^*} : G^* \to \mathcal{H}^*$, and hence φ^* is a weak embedding. This can be done by performing steps 0–3 on the graphs G and H, and the embedding ψ_φ, which will produce G^* and H^*, and an embedding $\psi_{\varphi^*} : G^* \to \mathcal{H}^*$. By construction, every 2-cut in H^*_u consists of a pair of clusters in $\partial\Delta_u$. Consequently, H^*_u has a unique embedding with the given outer cycle. In particular, $\widetilde{H_u}^*$ is planar. The rotation of the new clusters of H^* is uniquely defined and therefore must be consistent with any embedding of G, including ψ_φ.

Next, assume that φ^* is a weak embedding. Given an embedding $\psi_{\varphi^*} : G^* \to \mathcal{H}^*$, we construct an embedding $\psi_\varphi : G \to \mathcal{H}$ as follows. Let H^*_u be the subgraph of H^* induced by the clusters created by $\mathsf{clusterExpansion}(u)$. Note that H^* is a connected plane graph: the clusters created in step 0 are connected by a path (if $\deg(u) \leq 2$) or a cycle (if $\deg(u) \geq 3$); and any clusters created in step 3 (when $\deg(u) \geq 3$) are attached to this cycle. Since H^*_u is a connected plane graph, we may assume that there is a topological disk containing only the pipes and clusters of H^*_u; let D_u denote such a topological disk.

Let G^*_u be the subgraph of G^* mapped to H^*_u. We show that steps 0–3 of the operation can be reversed without introducing crossings. For each component C of G_u with pipe-deg$(C) \geq 3$, embed a wheel W_k in the disk D_{u_C} around the cluster u_C, and connect its external vertices to the vertices p_i, $i = 1, \ldots, k-1$, Since the pipes incident to u_C are empty, and each p_i is a unique vertex its cluster, this can be done without crossings. Now, every component C of G_u corresponds to a component C^* of G^*_u, and by (P1) every terminal vertex in C corresponds to terminal in C^*.

If we delete a component C^* from G^*, there will be a face F of D_u (a component of $D_u \setminus \psi_{\varphi^*}(G^*)$) that contains all terminals of C on its boundary. Denote by π_C the ccw cyclic order in which these terminals appear in the facial walk of F. If C admits an embedding in which the terminals appear in the outer face in the same order as π_C, we can embed C in F on the given terminals. We show that C admits such an embedding by proving that the SPQR trees of $\overline{C}$ and $\overline{C}^*$ impose the same

constraints on the cyclic order of terminals. Subdividing edges do not change these constraints. Step 1 does not change C. If step 2(b2) adds an edge in the skeleton of an S node, the possible combinatorial embeddings remain the same. Step 2(b3) maintains the rotation of v' and w'. If pipe-deg$(C) \geq 3$, C and C^* are identical apart from subdivided edges. Because all steps maintain the same constraints on the rotation system of the terminals, we can construct $\psi_\varphi : G \to \mathcal{H}$ by incrementally replacing the embedding of C^* by an embedding of C in D_u for every component C of G_u. By Lemma 3.2.2, this is possible without introducing crossings since no two components of G_u cross and every component is planar since φ is in simplified form.

Finally, we show that clusterExpansion(u) runs in $O(|E(G_u)| + \deg(u))$ time. Step 0 takes $O(\deg(u))$ time. Steps 1–3 are local operations that take $O(|E(C)|)$ time for each component C of G_u. Step 4 takes $O(|E(G_u)| + \deg(u))$ since each component C inserts at most $O(|E(C)|)$ pipes in H', where $|E(G_u)| = \sum_C |E(C)|$; the graph $\widetilde{H_u}$ has $\deg(u)$ more edges than H_u, and planarity testing takes linear time in the number of edges [HT74]. Step 5 takes $O(|E(G_u)|)$ time by Lemma 3.2.1 and (P1). □

3.3.2 Pipe Expansion

A cluster $u \in V(H)$ is called a *base of* an incident pipe $uv \in E(H)$ if every component of G_u is incident to: **(i)** at least one pipe-edge in uv; and **(ii)** at most one pipe-edge or exactly three pipe-edges that form a thick edge in any other pipe uw, $w \neq v$. We call a pipe uv *safe* if both of its endpoints are bases of uv; otherwise it is *unsafe*. For a simplified instance $\varphi : G \to H$ and a safe pipe $uv \in E(H)$, operation pipeExpansion(uv) consists of the following steps:

pipeExpansion(uv)**.** Input: an instance $\varphi : G \to H$ in simplified form and a safe pipe $uv \in E(H)$. The operation either reports that φ is not a weak embedding or returns an instance $\varphi' : G' \to H'$. First, produce an instance $\varphi^* : G \to H^*$ by contracting the pipe uv into the new cluster $\langle uv \rangle \in V(H^*)$ while mapping the vertices of G_u and G_v to $\langle uv \rangle$, see Fig. 3.5(top); and then apply simplify and clusterExpansion

to $\langle uv \rangle$.

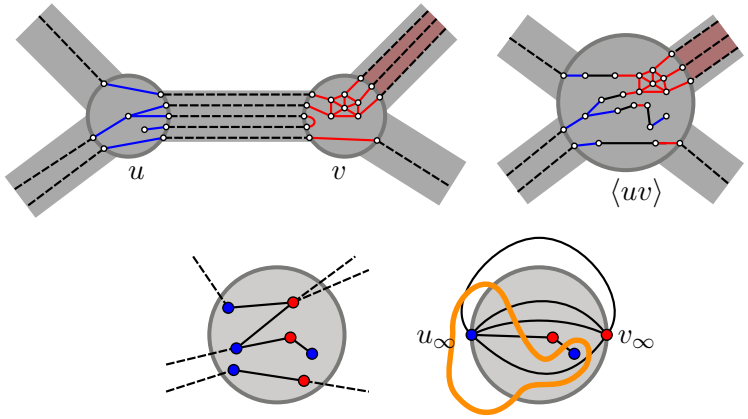


Figure 3.5: Pipe Expansion. A safe pipe uv (top left). The cluster $\langle uv \rangle$ obtained after contraction of uv (top right). The result of contracting the components in G_u and G_v (bottom left). The subsequent contraction of all components incident to $\partial_u D_{\langle uv \rangle}$ and $\partial_v D_{\langle uv \rangle}$, resp., and a Jordan curve that crosses every edge of the resulting bipartite plane multigraph (bottom right).

We use the following folklore result in the proof of correctness of operation pipeExpansion(uv). This result is obtained by an Euler tour algorithm on the dual graph of a plane bipartite multigraph.

Theorem 3.3.2 (Belyi [Bel83])**.** *For every embedded connected bipartite multigraph G^*, there exists a Jordan curve that crosses every edge of G^* precisely once. Such a curve can be computed in $O(|E(G^*)|)$ time.*

Lemma 3.3.3. *Given an instance $\varphi : G \to H$ and a safe pipe $uv \in E(H)$, pipeExpansion(uv) either reports that φ is not a weak embedding or produces an equivalent instance $\varphi' : G' \to H'$.*

Proof. Let $\varphi : G \to H$ be an instance in simplified form, and let uv be a safe pipe. Recall that pipeExpansion(uv) starts by producing an instance $\varphi^* : G \to H^*$ by contracting the pipe uv into the new cluster $\langle uv \rangle \in V(H^*)$ while mapping the vertices of G_u and G_v to $\langle uv \rangle$. It is enough to prove that φ and φ^* are equivalent, the rest of the proof follows from Lemmas 3.2.1 and 3.3.1.

One direction of the equivalence proof is trivial: Given an embedding $\psi_\varphi : G \to \mathcal{H}$, we can obtain an embedding $\psi_{\varphi^*} : G \to \mathcal{H}^*$ by defining $D_{\langle uv \rangle}$ as a topolo-

gical disk containing only D_u, D_v, and R_{uv}.

For the other direction, assume that we are given an embedding $\psi_{\varphi^*} : G \to \mathcal{H}^*$. We need to show that there exists an embedding $\psi_\varphi : G \to \mathcal{H}$. We shall apply Theorem 3.3.2 after contracting certain subgraphs of G_u and G_v (as described below). If a component contains cycles, then the contraction of its embedding creates a bouquet of loops. We study the cycles formed by G_u, G_v, and thick edges incident to G_u or G_v to ensure that no other component is embedded in the interior of such cycles.

Components of $G_{\langle uv\rangle}$ of pipe-degree 0. Note that the terminals corresponding to the pipes incident to u and v lie in two disjoint arcs of $\partial D_{\langle uv\rangle}$, which we denote by $\partial_u D_{\langle uv\rangle}$ and $\partial_v D_{\langle uv\rangle}$, respectively. The components of graph $G_{\langle uv\rangle}$ with positive pipe-degree are incident to terminals in $\partial_u D_{\langle uv\rangle}$ or $\partial_v D_{\langle uv\rangle}$ (possibly both). The components of pipe-degree 0 can be relocated to any face of the embedding of all other components. Without loss of generality, we may assume that all components of pipe-degree 0 lie in a common face incident to both G_u and G_v in ψ_{φ^*}.

Cycles induced by G_u and G_v, and by thick edges. Notice that (P3) and (P4) imply that every maximal biconnected component in G_u and G_v is a wheel. Since each wheel is 3-connected, the circular order of their external vertices is determined by the embedding ψ_{φ^*}. We may assume that no cycle in an embedding of a wheel subgraph encloses any vertex other than the center of the wheel. Indeed, suppose a 3-cycle (p_1, p_2, p_c) of a wheel encloses some vertex, where p_c is the center of the wheel, and p_1 and p_2 are two consecutive external vertices. We can modify the embedding of the edge p_1p_2 in ψ_{φ^*} so that it follows closely the path (p_1, p_c, p_2) and the 3-cycle does not contain any vertex; see Fig. 3.6(a).

Consider a thick edge θ in G between a wheel in G_u (or G_v) and a wheel in G_w for some adjacent cluster $w \notin \{u, v\}$. Recall that a thick edge consists of three paths, say $P_1 = (p_1, t_1, t_2, p_2)$, $P_2 = (p_3, t_3, t_4, p_4)$, and $P_3 = (p_5, t_5, t_5, p_6)$, where (p_1, p_3, p_5) and (p_2, p_4, p_6) are consecutive external vertices of the two wheels, resp., and p_i is the unique vertex in a cluster adjacent to terminal t_i for $i \in \{1, \ldots, 6\}$;

cf. (P1). If we suppress the terminals, then the two wheels incident to the thick edge would be in the same maximal 3-connected component of G and, therefore, their relative embedding is fixed. We can assume that no vertex is enclosed by the cycles induced by the vertices of the thick edge (i.e., by any pair of paths from P_1, P_2, and P_3). Indeed, we can modify the embedding of the path P_1 and P_3 in ψ_{φ^*} so that they closely follow the path $p_1p_3 \cup P_2 \cup p_4p_2$ and $p_5p_3 \cup P_2 \cup p_4p_6$, respectively. By (P4), such modification of the embedding is always possible without introducing crossings; see Fig. 3.6(b). We conclude that a cycle induced by the thick edge θ does not enclose any vertices of G.

Separating G_u and G_v. We next show that there exits a closed Jordan curve that separates the embeddings of G_u and G_v, and crosses every edge between G_u and G_v precisely once. In order to use Theorem 3.3.2, we reduce $G_{\langle uv\rangle}$ and its embedding to a bipartite multigraph in two steps.

(1) Contract each component C of G_u (resp., G_v) to a single vertex w_C. This results in a bouquet of loops at w_C. As argued above, a cycle through the external vertices of a wheel in G_u or G_v encloses only its center. Hence, none of the loops at w_C encloses any other vertex of G, and they can be discarded.

(2) As uv is safe, every component C of G_u and G_v is incident to either at most one terminal in $\partial D_{\langle uv\rangle}$ or precisely three terminals corresponding to a thick edge. Contract the arc $\partial_u D_{\langle uv\rangle}$ (resp., $\partial_v D_{\langle uv\rangle}$) and for every component C of G_u (resp.,

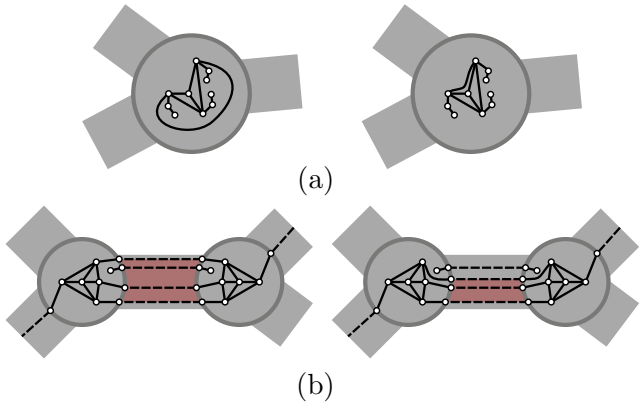


Figure 3.6: Changing the embedding so that: (a) no cycle encloses vertices that are not the center of a wheel; and (b) no induced cycle of a thick edge encloses a vertex.

G_v) all edges between w_C and terminals into a new vertex $u_\infty \in \partial D_{\langle uv \rangle}$ (resp., $v_\infty \in \partial D_{\langle uv \rangle}$), see Fig. 3.5(bottom-left); and insert an edge $u_\infty v_\infty$ in the exterior of $D_{\langle uv \rangle}$. This results in a bouquet of loops at u_∞ (resp., v_∞), corresponding to thick edges. As argued above, these loops do not enclose any vertices, and can be eliminated. The subgraph of all components of pipe-degree 1 or higher has been transformed into a connected bipartite plane multigraph, and each component of pipe-degree 0 is also transformed into a connected bipartite plane multigraph.

We apply Theorem 3.3.2 for each of these plane multigraphs independently. By our assumption, the components C, pipe-deg$(C) = 0$, lie in a common face. Consequently, we can combine their Jordan curves with the Jordan curve of all remaining components into a single Jordan curve that crosses every edge between G_u and G_v precisely once, see Fig. 3.5(bottom-right). After reversing the contractions and loop deletions described above, we obtain a Jordan curve γ that separates G_u and G_v, and crosses every edge between G_u and G_v precisely once.

Since γ crosses the edge $u_\infty v_\infty$ precisely once, and $u_\infty v_\infty$ is homotopic to both arcs of $\partial D_{\langle uv \rangle}$ between u_∞ and v_∞, we can assume that γ crosses each of these arcs precisely once. Consequently, the Jordan arc $\gamma' = \gamma \cap D_{\langle uv \rangle}$ partitions the disk $D_{\langle uv \rangle}$ into two disks, D_u and D_v, adjacent to $\partial_u D_{\langle uv \rangle}$ and $\partial_v D_{\langle uv \rangle}$, respectively, such that every edge between G_u and G_v crosses γ' precisely once and all other edges of $G_{\langle uv \rangle}$ lie entirely in either D_u or D_v. This yields the required embedding $\psi_\varphi : G \to \mathcal{H}$. □

3.4 Algorithm and Runtime Analyis

In this section, we present our algorithm for recognizing weak embeddings. We describe the algorithm and prove that it recognizes weak embeddings in Section 3.4.1. A naïve implementation would take $O(m^2)$ time as explained below; we describe how to implement it in $O(m \log m)$ time using additional data structures in Section 3.4.3.

3.4.1 Main Algorithm

We are given a piecewise linear simplicial map $\varphi : G \to H$, where G is a graph and H is an embedded graph in an orientable surface. We introduce some terminology for an instance $\varphi : H \to G$.

- A component C of G_u, in a cluster $u \in V(H)$, is *stable* if pipe-deg$(C) = 2$ and, in each of the two pipes, C is incident to exactly one edge or exactly one thick edge. Otherwise C is *unstable*.
- Recall that a pipe $uv \in E(H)$ is *safe* if both u and v are bases of uv; otherwise it is *unsafe*.
- A cluster $u \in V(H)$ (resp., a pipe $uv \in E(H)$) is *empty* if $\varphi^{-1}[u] = \emptyset$ (resp., $\varphi^{-1}[uv] = \emptyset$). Otherwise u is *nonempty*.
- A safe pipe $uv \in E(H)$ is *useless* if uv is empty or if u and v are each incident to exactly two nonempty pipes and every component of G_u and G_v is stable. The safe pipe uv is *useful* otherwise (i.e., if uv is nonempty; and u or v is incident to at least 3 nonempty pipes, or G_u or G_v has an unstable component).

Notice that the pipe expansion of a useless pipe may not change the instance combinatorially. As we show below (Lemma 3.4.6), our algorithm reduces G to a collection of *thick cycles*, defined as a cycle in which each node is a component of G_u for some cluster u, and each edge is either an edge or a thick edge between wheels. (In particular, if we contract every wheel to a single vertex, then a thick cycle reduces to a cycle, possibly with multiple edges coming from thick edges.)

Data Structures. The graph G is stored using adjacency lists. We store the combinatorial embedding of H using the rotation system of H (ccw order of pipes incident to each $u \in V(H)$). The mapping φ is encoded by its restriction to $V(G)$: that is, by the images $\varphi(p)$ for all $p \in V(G)$. For each cluster $u \in V(H)$, we store the set of components of G_u, each stored as the ID of a representative edge and the pipe-degree of the component. Every pipe $uv \in E(H)$ has two Boolean variables

to indicate whether the respective endpoints are its bases; and an additional Boolean variable indicates whether uv is useful. These data structures are maintained dynamically as the algorithm modifies H and G.

Algorithm(φ)**.** Input: an instance $\varphi : G \to H$ in simplified form.

Phase 1. Apply clusterExpansion to each $u \in V(H)$. Denote the resulting instance by $\varphi' : G' \to H'$. Build the data structures described above for $\varphi' : G' \to H'$.

Phase 2. While there is a useful pipe in H', let $uv \in E(H')$ be an arbitrary useful pipe and apply pipeExpansion(uv).

Phase 3. If any component of G' that contains a wheel is nonplanar, then report that φ is not a weak embedding and halt. Otherwise, in each cluster contract every wheel component to a single vertex, and turn every thick edge into a single edge by removing multiple edges. Denote the resulting instance by $\varphi'' : G'' \to H''$. If any component C of G'' is a cycle with k vertices but $\varphi''(C)$ is not a cycle with k clusters in H'', then report that φ is not a weak embedding, else report that φ is a weak embedding. This completes the algorithm.

3.4.2 Analysis of Algorithm

We show that the Algorithm recognizes whether the input $\varphi : G \to H$ is a weak embedding. The running time analysis follows in Section 3.4.3.

Termination. Since both Phase 1 and Phase 3 consist of for-loops only, it is enough to show that the while loop in Phase 2 terminates. We define a nonnegative potential function $\Phi_1(\varphi)$ for an instance $\varphi : G \to H$. For a pipe $uv \in E(H)$, let $\sigma(uv)$ be the number of pipe-edges in $\varphi^{-1}[uv]$ minus twice the number of thick edges (so that each thick edge is counted as one). Let $s(u)$ be the number of stable components of G_u. We now define the following quantities for an instance

$\varphi : G \to H$:

$$N_\sigma(\varphi) = \sum_{uv \in E(H)} \sigma(uv),$$
$$N_s(\varphi) = \sum_{u \in V(H)} s(u). \tag{3.1}$$

Let $Q(\varphi)$ be the number of nonempty pipes of an instance φ. We can now define the potential function

$$\Phi_1(\varphi) = 4N_\sigma(\varphi) + N_s(\varphi) - 4Q(\varphi). \tag{3.2}$$

Note that $\Phi_1(\varphi)$ is a nonnegative integer since $N_\sigma(\varphi) \geq Q(\varphi)$.

The following lemma describes the effect of one iteration of the while loop in Phase 2 on Φ_1 and $N_\sigma - Q$.

Lemma 3.4.1. *Assume we invoke* pipeExpansion(uv) *for a useful pipe uv in an instance φ, obtaining instance φ'. Then,*

- $\Phi_1(\varphi) > \Phi_1(\varphi')$, *and*
- $N_\sigma(\varphi) - Q(\varphi) \geq N_\sigma(\varphi') - Q(\varphi')$.

Proof. Let C be a component of $G_{\langle uv \rangle}$. Let $\sigma_C(uv)$ be the number of pipe-edges of C in $\varphi^{-1}[uv]$ minus twice the number of thick edges of C in $\varphi^{-1}[uv]$ before pipeExpansion(uv). Let s_C be the number of stable components of G_u and G_v contained in C before pipeExpansion(uv). Then

$$\sigma(uv) = \sum_C \sigma_C(uv) \qquad \text{and} \qquad s(u) + s(v) = \sum_C s_C,$$

where the summation is over all components C of $G_{\langle uv \rangle}$. We distinguish cases based on the pipe-degree of C.

Assume pipe-deg$(C) = 1$. Then $\sigma_C(uv) \geq 1$ by the definition of safe pipes. Step 1 of pipeExpansion creates component C' of pipe-degree 1 in some cluster on the boundary of $\Delta_{\langle uv \rangle}$ and no new pipe-edge in $\Delta_{\langle uv \rangle}$. Consequently, pipeExpansion

decreases the contribution of C to N_σ and N_s to 0. The contribution of C to $N_\sigma - Q$ does not decrease (i.e., remains constant 0) only if $\sigma_C(uv) = 1$ and C is the only component in $G_{\langle uv \rangle}$. In this case, however, C contains precisely one component in each of G_u and G_v, of pipe-degree 1 and 2. The component of pipe-degree 2 is stable since uv is safe and $\sigma_C(uv) = 1$, and so the number of stable components decreases by one, hence pipeExpansion decreases the contribution of C to Φ_1.

Assume pipe-deg$(C) = 2$ (see Fig. 3.2). Then Step 2 of pipeExpansion creates two stable components, connected by either a single edge or a thick edge in a pipe in $\Delta_{\langle uv \rangle}$. Consequently, pipeExpansion(uv)) changes the contribution of C to N_σ from at least 1 to precisely 1. The number of stable components increases if $s_C \in \{0, 1\}$. Then, at least one component of G_u and G_v in C is unstable. Since uv is useful, this component is incident to at least two single or thick edges in uv, and so $\sigma_C(uv) \geq 2$. In this case, the contribution of C to N_σ decreases by at least one, while the number of stable components increases by at most two, hence Φ_1 strictly decreases. The contribution of C to Φ_1 is unchanged only if C contains precisely one component in each of G_u and G_v, where $\sigma_C(uv) = 1$ and $s_C = 2$ (that is, both are stable components).

Assume pipe-deg$(C) \geq 3$. Let W_k, $k \geq 4$, be the wheel created by clusterExpansion(uv). Since uv is a safe pipe, every component of G_u or G_v contained in C is incident to at most one (possibly thick) pipe-edge outside of $\varphi^{-1}[uv]$. Hence, C contains at least $k - 2$ edges between these components of G_u and G_v (which are pipe-edges in uv), and its contribution to N_σ is at least $k - 2$. Step 3 of pipeExpansion(uv) creates $k - 1$ components in distinct clusters of $\Delta_{\langle uv \rangle}$. Hence, the contribution of C to N_s increases by at most $k - 1$. Step 3 also creates $k - 1$ pipe-edges, each of which is the only pipe-edges in its pipe, hence they contribute zero to $N_\sigma - Q$. The difference between the new and the old value of $N_\sigma - Q$ due to C is at most $-(k - 2 - 1) = -k + 3$ (where the term -1 accounts for the case that C is the only component of $G_{\langle uv \rangle}$). Hence, we obtain the following upper bound on the difference between the new and the old value of Φ_1 accounted to C: $-4(k - 3) + (k - 1) = 11 - 3k \leq 11 - 3 \cdot 4 = -1 < 0$, where the $(k - 1)$-term

corresponds to the increase in N_s. Consequently the contribution of C to both, Φ_1 and $N_\sigma - Q$, decreases.

Summing over the contributions of all components of $G_{\langle uv \rangle}$, we have shown that neither Φ_1 nor $N_\sigma - Q$ increases. By the previous case analysis, the contribution of every component C to Φ_1 and $N_\sigma - Q$ does not increase. The contribution of C to Φ_1 is unchanged when C contains exactly one component in each G_u and G_v, and both are stable, and hence the contribution of C to N_s is remains 2. For contradiction, assume that Φ_1 and $N_\sigma - Q$ are both unchanged. Then all components of $G_{\langle uv \rangle}$ are stable, and each contains precisely one (stable) component in G_u and G_v, respectively. Then N_s and N_σ remain unchanged. Therefore, Q must also remain unchanged, which implies that all components of $G_{\langle uv \rangle}$ are adjacent to the same pair of pipes. Then, uv is useless, a contradiction.

□

Corollary 3.4.2. *The algorithm executes at most* $15m$ *iterations of Phase 2 and, therefore, terminates; and it creates at most* $12m$ *stable components.*

Proof. At the end of Phase 1, there are at most $3m$ pipe-edges: We charge the creation of pipe-deg(C) pipe-edges in new pipes in Step 3 of cluster expansion to pipe-deg(C) original pipe-edges incident to C. Similarly we charge the creation of a (thick) pipe-edge created in Step 2 to 1 or 3 (in the case we created a thick edge) original pipe-edges incident to C. Clearly, each original edge receives the charge of at most 2, one from each of its incident components. Hence, in total we introduced at most $2m$ new pipe-edges.

By definition $N_\sigma - Q$ is a nonnegative integer, and its initial value does not exceed the number of pipe-edges. Therefore, at the end of Phase 1, we have $0 \leq N_\sigma - Q \leq 3m$. Since each pipe-edge can be incident to at most two stable components and each stable component must be incident to two pipe-edges, $0 \leq N_s \leq 3m$. Hence, $0 \leq \Phi_1 \leq 15m$.

By Lemma 3.4.1, Φ_1 strictly decreases in each iteration of the while loop in Phase 2. Therefore Phase 2 has at most $15m$ iterations. Since both Phase 1 and

Phase 3 consist of for-loops only, the algorithm terminates.

By Lemma 3.4.1, Φ_1 strictly decreases and $N_\sigma - Q$ does not increase in each iteration of the while loop in Phase 2. Consequently, we can charge the creation of a stable component to the decrease of one fourth of a unit of $N_\sigma - Q$. Hence, Phase 2 creates at most $12m$ stable components. □

Correctness. We show next that Phase 2 reduces G' to a collection of thick cycles (see Lemma 3.4.6 below). First, we need a few observations.

Lemma 3.4.3. *Every cluster u' created by an operation* clusterExpansion(u) *satisfies the following:*

(B1) u' is either empty or the base for some nonempty pipe.

(B2) If u' lies in the interior of the disk Δ_u, then u' is either empty or the base of a unique nonempty pipe $u'v'$, where v' in on the boundary of Δ_u.

(B3) If u' lies on the boundary of the disk Δ_u, then u' is either empty, or the base of at least one and at most two pipes, exactly one of which is outside of Δ_u.

(B4) If u' is nonempty, then u' is incident to at most three empty pipes.

Proof. Operation clusterExpansion(u) creates a cluster u_v on the boundary of the disk Δ_u for every pipe uv incident to u. By construction, u_v is either empty (if the pipe uv was empty), or a base for the pipe vu_v, which lies outside of Δ_u. When u_v is nonempty, the only incident pipes that could be empty are created in Step 0 (either two pipes of the cycle C_u or a single pipe if $\deg(u) = 2$). For each component C of G_u with pipe-deg$(C) \geq 3$, clusterExpansion(u) creates a wheel where the center cluster is empty and every external cluster is incident to exactly one nonempty pipe (to a cluster on the boundary of Δ_u), consequently, it is a base for that pipe. The external clusters are incident to exactly three empty pipes by (P4). By construction, clusters in $\partial\Delta_u$ are incident to at most two empty pipes. □

Corollary 3.4.4. *In every step of Phase 2, every cluster $u \in V(H')$ is either empty or the base for at least one nonempty pipe.*

Proof. Phase 1 of the algorithm performs clusterExpansion(u) for every cluster of the input independently. At the end of Phase 1 (i.e., beginning of Phase 2), each cluster in H' has been created by a clusterExpansion operation, and (B1) follows from Lemma 3.4.3.

Subsequent steps of Phase 2 successively apply pipeExpansion operations thereby creating new clusters. By Lemma 3.4.3, property (B1) is established for all new clusters, and it continues to hold for existing clusters. □

Lemma 3.4.5. *In every step of Phase 2, if H' contains a nonempty unsafe pipe, then it also contains a useful pipe.*

Proof. Let u_0u_1 be a nonempty unsafe pipe in H'. Without loss of generality, assume that u_1 is not a base for u_0u_1. We iteratively define a simple path $(u_0, u_1, \ldots, u_\ell)$ for some $\ell \in \mathbb{N}$ as follows. Assume that $i \geq 1$, vertices $u_0, \ldots, u_i$ have been defined, and $u_{i-1}u_i$ is a nonempty unsafe pipe in H' for which u_i is not a base. By (B1), u_i is a base for some nonempty pipe u_iw, where $w \neq u_{i-1}$. Put $u_{i+1} = w$. This iterative process terminates when either u_iu_{i+1} is a nonempty safe pipe, or $u_{i+1} = u_k$ for some $0 \leq k < i-1$.

Case 1: The iterative process finds a nonempty safe pipe u_iu_{i+1}. We claim that u_iu_{i+1} is useful. Suppose, to the contrary, that u_iu_{i+1} is useless. Then u_i is incident to two nonempty pipes (which are necessarily $u_{i-1}u_i$ and u_iu_{i+1}), and every component in G_{u_i} has pipe-degree 2. Consequently, u_i is a base for both $u_{i-1}u_i$ and u_iu_{i+1}. This contradicts our assumption that u_i is not a base for $u_{i-1}u_i$; and proves the claim.

Case 2: The iterative process finds a cycle $U = (u_k, u_{k+1}, \ldots, u_\ell)$ where $u_{\ell+1} = u_k$ and for every $i = k, \ldots, \ell$, u_i is a base for u_iu_{i+1} but not a base for u_iu_{i-1}. We show that this case does not occur. All clusters in the cycle have been created by clusterExpansion operations (in Phase 1 or 2). We claim that not all clusters in U are created by the same clusterExpansion operation. For otherwise, by (B2) u_i and u_{i+1}, for some i, are on the boundary of an expansion disk Δ_u then u_{i+1} is not the base of u_iu_{i+1} by the construction of U. Furthermore, u_iu_{i+1} is

nonempty by the definition of a base. By (B3), u_{i+1} is the base of a single pipe, which is outside of Δ_u, as the other base would have to be $u_i u_{i+1}$. Due to the previous claim and since U is a cycle, there are two consecutive clusters, u_i and u_{i+1}, in U such that u_i was created by an earlier invocation of clusterExpansion than u_{i+1}. By (B3), u_{i+1} is a base for $u_i u_{i+1}$. This contradicts our assumption, and proves that Case 2 does not occur. □

Lemma 3.4.6. *The following hold for the instance $\varphi' : G' \to H'$ at the end of Phase 2:*

1. *every pipe in $E(H')$ is empty or useless,*
2. *every component of G' is a thick cycle, and*
3. *any two components of G' are mapped to the same or two vertex-disjoint cycles in H'.*

Proof. **1**. When the while loop of Phase 2 terminates, there are no useful pipes in H'. Consequently, every safe pipe is useless. By Lemma 3.4.5, every unsafe pipe is empty at that time. Overall, every pipe is empty or useless, as claimed.

2. Since every pipe $uv \in E(H')$ is empty or useless, every component in every cluster is stable. It follows that every component of G' must be a thick cycle.

3. Since every pipe $uv \in E(H')$ is empty or useless, every cluster is incident to at most two nonempty pipes. Consequently, any two thick cycles of G' are mapped to either the same cycle or two disjoint cycles in H'. □

Lemma 3.4.7. *The algorithm reports whether φ is a weak embedding.*

Proof. By Lemmas 3.3.1 and 3.3.3, every operation either reports that the instance is not a weak embedding and halts, or produces an instance equivalent to the input φ. Consequently, if any operation finds a negative instance, then φ is not a weak embedding. Otherwise the while loop in Phase 2 terminates, and yields an instance $\varphi' : G' \to H'$ equivalent to φ. By Lemma 3.4.6, every component of G' is a thick cycle, any two of which are mapped to the same or vertex-disjoint cycles in H'.

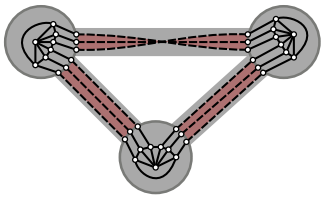


Figure 3.7: A nonplanar thick cycle that does not embed in an annulus. It embeds in the Möbius band or the projective plane.

The strip system of the subgraph $\varphi'(C)$ of H', where C is a thick cycle, is homeomorphic to the annulus, since M is orientable. If C is nonplanar, then φ' is clearly not a weak embedding (see Fig. 3.7). Furthermore, a thick cycle C that winds around $\varphi'(C)$ several times cannot be embedded in that annulus (Fig. 3.1(d)). However, one or more thick cycles that each wind around $\varphi'(C)$ once can be embedded in nested annuli in the strip system $\mathcal{H}'$ (Fig. 3.1(c)). This completes the proof of correctness of Algorithm(φ). □

3.4.3 Efficient Implementation

Recall that pipeExpansion(uv) first contracts the pipe uv into the new cluster $\langle uv \rangle \in V(H^*)$. Each stable component C of $G_{\langle uv \rangle}$ is composed of two stable components, in G_u and G_v respectively. Then clusterExpansion(uv) performed at the end of pipeExpansion(uv) splits C into two stable components in two new clusters. That is, two adjacent stable components in G_u and G_v are replaced by two adjacent stable components in two new clusters. Consequently, we cannot afford to spend $O(|E_{\langle uv \rangle}|)$ time for pipeExpansion(uv). We introduce auxiliary data structures to handle stable components efficiently: we use *set operations* to maintain a largest set of stable components in $O(1)$ time. A dynamic variant of the heavy path decomposition yields an $O(m \log m)$ bound on the total time spent on stable components in the main loop of the algorithm.

Data structures for stable components. For each pair (u, uv) of a cluster $u \in V(H)$ and an incident pipe $uv \in E(H)$, we store a set $L(u, uv)$ of all stable components of G_u adjacent to a (thick) edge in $\varphi^{-1}[uv]$. For every cluster $u \in V(H)$,

let $w^*(u)$ be a neighbor of u maximizing the size of $L(u, uw^*(u))$; we maintain a pointer from u to a set $L(u, uw^*(u))$. The total number of sets $L(u, uv)$ and the sum of their sizes are $O(m)$. We can initialize them in $O(m)$ time. For each cluster $u \in V(H)$, we store the components of G_u in two sets: a set of stable and unstable components, respectively, each stored as the ID of a representative edge of the component. Hence, for each pipe we can determine in $O(1)$ time whether uv is useful or useless. Every stable component in G_u has a pointer to the two sets $L(u, .)$ in which it appears.

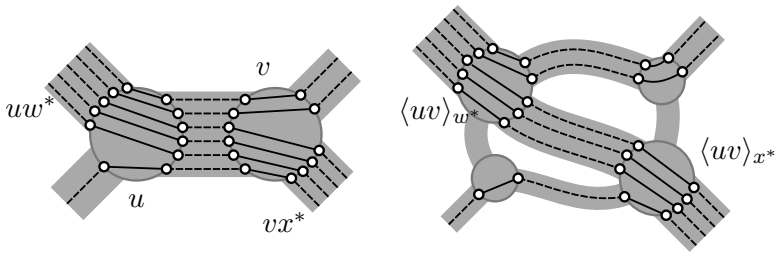


Figure 3.8: Implementation of pipeExpansion(uv) for stable components.

In each iteration of Phase 2, we implement pipeExpansion(uv) for a useful pipe uv as follows. Let $G^{\diamond}_{\langle uv \rangle}$ be the graph formed by the unstable components of $G_{\langle uv \rangle}$ and stable components that are not in simplified form, i.e., stable components C where $\sigma_C(uv) > 1$. Compute $G^{\diamond}_{\langle uv \rangle}$ using DFS starting from each unstable component of G_u and G_v. If a stable component of G_u and G_v is absorbed by a new unstable component of $G_{\langle uv \rangle}$, delete it from the set $L(u, .)$ or $L(v, .)$ in which it appears.

(a) We perform clusterExpansion only on $G^{\diamond}_{\langle uv \rangle}$.

(b) We handle the remaining stable components of $G_{\langle uv \rangle}$ as follows (refer to Fig. 3.8). Since uv is safe, every stable component of G_u and G_v appears in $L(u, uv)$ and $L(v, vu)$, respectively. For all w, $w \notin \{v, w^*\}$, where $w^* = w^*(u)$, move all components from $L(u, uw)$ to the cluster $\langle uv \rangle_w$. Similarly, for all x, $x \notin \{u, x^*\}$, where $x^* = w^*(v)$, move all components from $L(v, vx)$ to the cluster $\langle uv \rangle_x$. Let $L'(u, uv)$ and $L'(v, uv)$ be the sets obtained from $L(u, uv)$ and $L(v, uv)$ after this step. Notice that the remaining stable components of G_u and G_v in $L'(u, uv)$ and $L'(v, uv)$ ap-

pear in $L(u, uw^*)$ and $L(v, vx^*)$, respectively. Move them to the clusters $\langle uv\rangle_{w^*}$ and $\langle uv\rangle_{x^*}$ by renaming all data structures of u and v, respectively, which can be done in $O(1)$ time. Update H so that it reflects the changes in G, by creating appropriate pipes between the clusters in $\partial\Delta_{\langle uv\rangle}$, checking for crossings by local planarity testing inside $\Delta_{\langle uv\rangle}$ (similar to Step 4 of clusterExpansion).

(c) We add the new stable components (created by clusterExpansion from unstable components) that fit the definition of $L(\langle u, v\rangle_{w^*}, \langle u, v\rangle_{w^*}\langle u, v\rangle_{x^*})$ to the current set $L'(u, uv)$ to obtain the set $L(\langle u, v\rangle_{w^*}, \langle u, v\rangle_{w^*}\langle u, v\rangle_{x^*})$. Analogously, add new stable components to the current sets $L'(v, vu)$, $L(u, uw)$ where $w \neq v$, and $L(v, vx)$ where $x \neq u$ to obtain the new sets $L(\langle u, v\rangle_{x^*}, \langle u, v\rangle_{x^*}\langle u, v\rangle_{w^*})$, $L(\langle u, v\rangle_w, \langle u, v\rangle_w w)$, and $L(\langle u, v\rangle_x, \langle u, v\rangle_x x)$, respectively. Compute any other sets of stable components from scratch.

We are now ready to show that the running time of the algorithm improves to $O(m \log m)$.

Lemma 3.4.8. *Our implementation of the algorithm runs in $O(m \log m)$ time.*

Proof. Phases 1 and 3 take $O(m)$ time by Lemmas 3.4.1, 3.3.1, and 3.4.6. The while loop in Phase 2 terminates after $O(m)$ iterations by Corollary 3.4.2. Using just Lemmas 3.2.1 and 3.3.1, each iteration of Phase 2 would take $O(m)$ time leading to an overall running time of $O(m^2)$. We define a new potential function for an instance $\varphi : G \to H$ to show that each iteration of Phase 2 takes $O(\log m)$ amortized time.

For every $u \in V(H)$, let $\mathcal{L}(u)$ be the number of stable components in G_u. Let s be the number of stable components created from the beginning of Phase 2 up to the current iteration. We define a new potential function as[1]

$$\Phi_2(\varphi) = \Phi_1(\varphi) + (12m - s)\log(28m) + \sum_{u \in V(H)} \mathcal{L}(u) \log \mathcal{L}(u).$$

By Corollary 3.4.2 the second term is nonnegative. Note that $\Phi_2(\varphi) = O(m \log m)$ since $\Phi_1(\varphi) = O(m)$ and $\sum_{u \in V(H)} \mathcal{L}(u) = O(m)$. We show that Φ_2 strictly decreases

[1] All logarithms are of base 2.

in Phase 2. As argued above (cf. Lemma 3.4.1), Φ_1 strictly decreases. The second term of Φ_2 can only decrease since s increments when stable components are created (but never decrements). The term $\sum_{u \in V(H)} \mathcal{L}(u) \log \mathcal{L}(u)$ increases when new stable components are created. However, this increase is offset by the decrease in the second term of Φ_2. It suffices to consider the case that $\mathcal{L}(u)$ increments from k to $k+1$. Then

$$\begin{aligned} k \log k + \log(28m) &\geq \left(k \log(k+1) - k \log \frac{k+1}{k} \right) + (2 + \log(k+1)) \\ &\geq (k+1) \log(k+1) + 2 - \log \left(1 + \frac{1}{k} \right)^k \\ &\geq (k+1) \log(k+1) + 2 - \log e \\ &> (k+1) \log(k+1), \end{aligned}$$

that is, the decrease of $\log(28m)$ in the second term offsets the increase of $(k+1)\log(k+1) - k \log k$ in the third term.

We next show that the time spent on each iteration of the while loop in Phase 2 is bounded above by a constant times the decrease of the potential Φ_2. This will complete the proof since the potential Φ_2 is nonincreasing throughout the execution of the algorithm as we have just shown.

For a useful pipe $uv \in E(H)$, let $\lambda(uv) = \Lambda$ be the collection of sets $L(u, uw)$ where $w \notin \{v, w^*(u)\}$ and $L(v, vx)$ where $x \notin \{u, w^*(v)\}$. Our implementation of $\mathsf{pipeExpansion}(uv)$ spends $O(|E(G^\diamond_{\langle uv \rangle})| + \deg(\langle uv \rangle))$ time to process the components of $G^\diamond_{\langle uv \rangle}$, by Lemma 3.3.1, and $O(1 + \sum_{L \in \Lambda} |L|)$ to process the remaining stable components. However, by (B4), $\deg(\langle uv \rangle) = O(|E(G^\diamond_{\langle uv \rangle})| + 1 + \sum_{L \in \Lambda} |L|)$. Hence the running time of $\mathsf{pipeExpansion}(uv)$ is

$$O\left(|E(G^\diamond_{\langle uv \rangle})| + 1 + \sum_{L \in \Lambda} |L| \right). \tag{3.3}$$

First, let C be a component of $G^\diamond_{\langle uv \rangle}$. We have seen (in the proof of Lemma 3.4.1) that $\mathsf{pipeExpansion}(uv)$ decreases N_σ by at least $\sigma_C(uv) - 1$ due to edges in C. By the

definition of safe pipes, C contains a nonempty set A of components of G_u and a nonempty set B of components of G_v. By Lemma 3.2.4, each component $C' \in A \cup B$ contains $O(t_{C'})$ edges, where $t_{C'}$ is the number of terminals of C'. Notice that $\sigma_C(uv) \geq \sum_{C' \in A}(t_{C'} - 3)$, because at most three out of $t_{C'}$ terminals are not adjacent to a pipe-edge in $\varphi^{-1}[uv]$ by the definition of safe pipes (equality occurs when a thick edge is incident to C' in a pipe other than uv). As such, for a component C in $G^{\diamond}_{\langle uv \rangle}$, the contribution of C to $N_\sigma - Q$ decreases by $\Omega(|E(C)|)$. Therefore, the first term of (3.3) is charged to the decrease in $N_\sigma - Q$. By Lemma 3.4.1, summation over pipe expansions in Phase 2 yields $O(m)$. This takes care of steps **(a)** and **(c)** of the efficient implementation (Section 3.4.3). It remains to bound the time complexity of step **(b)**, which deals exclusively with stable components.

Second, we show that the time that $\mathsf{pipeExpansion}(uv)$ spends on stable components is absorbed by the decrease in the last term of Φ_2. When we move the components of $L(u, uw)$, $w \notin \{v, w^*\}$, to the cluster $\langle uv \rangle_w$, we spend linear time on all but a maximal set, which can be moved in $O(1)$ time using a set operation. In what follows we show that a constant times the corresponding decrease in the term $\sum_{u \in V(H)} \mathcal{L}(u) \log \mathcal{L}(u)$ subsumes this work.

We adapt the analysis from the classic heavy path decomposition. Suppose we partition a set of size $k = \mathcal{L}(u) = \mathcal{L}(v)$ into ℓ subsets of sizes $k_1 \geq \ldots \geq k_\ell$. Note that $k_j \leq k/2$ for $j \geq 2$. Then

$$k \log k = \sum_{i=1}^{\ell} k_i \log k \geq k_1 \log k_1 + \sum_{j=2}^{\ell} k_j \log(2k_j) = \sum_{i=1}^{\ell} k_i \log k_i + \sum_{j=2}^{\ell} k_j.$$

Hence, the decrease in $\sum_{u \in V(H)} \mathcal{L}(u) \log \mathcal{L}(u)$, which is equal to $k \log k - \sum_{i=1}^{\ell} k_i \log k_i$, is bounded from below by $k - k_1$. Therefore if we spend $O(1)$ time on a maximal subset of size k_1, we can afford to spend linear time on all other subsets. Thus, the decrease in $\sum_{u \in V(H)} \mathcal{L}(u) \log \mathcal{L}(u)$ subsumes the actual work and this concludes the proof. □

This completes the proof of Theorem 3.1.1(i). Part (ii) of Theorem 3.1.1 is shown in Section 3.5.

3.5 Constructing an embedding

Our recognition algorithm in Section 3.4 decides in $O(m \log m)$ time whether a given instance φ is a weak embedding. However, if φ turns out to be a weak embedding, it does not provide an embedding ψ_φ, since at the end of the algorithm we have an equivalent "reduced" instance φ' at hand. In this section, we show how to compute the combinatorial representation of an embedding ψ_φ for the input φ.

Assume that $\varphi : G \to H$ is a weak embedding. By Lemmas 3.4.7 and 3.4.8, we can obtain a combinatorial representation of an embedding $\pi_{\varphi'} \in \Pi(\varphi')$ in $O(n \log n)$ time of the instance $\varphi' : G' \to H'$ produced by the algorithm at the end of Phase 2.

We sequentially reverse the steps of the algorithm, and maintain a combinatorial embeddings for all intermediate instances until we obtain a combinatorial representation $\pi_\varphi \in \Pi(\varphi)$. By Lemma 3.2.2, we can then obtain an embedding $\psi_\varphi : G \to \mathcal{H}$ in $O(m)$ time. Reversing a $\mathsf{clusterExpansion}(u)$ operation is trivial: the total orders of pipes in Δ_u can be ignored and the total order for the pipes uv are the same as orders for $u_v v$. This can be done in $O(\deg(u))$ time.

Let $\varphi^{(1)} : G^{(1)} \to H^{(1)}$ be the input instance of $\mathsf{pipeExpansion}(uv)$, $\varphi^{(2)} : G^{(1)} \to H^{(2)}$ be the instance obtained by contracting uv and $\varphi^{(3)} : G^{(3)} \to H^{(3)}$ be the instance after $\mathsf{clusterExpansion}(\langle uv \rangle)$. By the previous argument, the total orders of pipe-edges $\pi_{\varphi^{(2)}}(\langle uv \rangle w)$ of all pipes $\langle uv \rangle w$ can be obtained from a combinatorial representation $\pi_{\varphi^{(3)}}$ in $O(\deg(\langle uv \rangle))$ time. These orders also correspond to $\pi_{\varphi^{(1)}}(uw)$ and $\pi_{\varphi^{(1)}}(vx)$ for pipes uw and vx in $\varphi^{(1)}$ where $w \neq v$ and $x \neq u$.

To obtain an order $\pi_{\varphi^{(1)}}(uv)$, we embed $G^{(1)}_{\langle uv \rangle}$ into $D_{\langle uv \rangle}$ using Lemma 3.2.2 in $O(|E(G^{(1)}_{\langle uv \rangle})|)$ time and find the Jordan curve defined in the proof of Lemma 3.3.3. The order $\pi_{\varphi^{(1)}}(uv)$ of the pipe-edges in $\varphi^{-1}[uv]$ is given by the order in which the Jordan curve intersects these edges. This takes $O(|E(G^{(1)}_{\langle uv \rangle})|)$ time by Theo-

rem 3.3.2, though we first need to obtain an embedding of $G^{(1)}$, where triangles of wheels in $G^{(1)}_{\langle uv\rangle}$ and 4-cycles induced by thick edges in and incident to $G^{(1)}_{\langle uv\rangle}$ are empty. This can be done by changing the rotation at the vertices of the wheels and 4-cycles corresponding to thick edges one by one in $O(|E(G^{(1)}_{\langle uv\rangle})|)$ time, since uv is safe in the resulting instance.

However, this would lead to a $O(m^2)$ worst case time complexity because of stable components. We show how to reduce the running time to $O(m \log m)$. Let us call a pipe-edge (thick edge) *stable* if it connects two stable components in two adjacent clusters. In each total order $\pi_{\varphi^*}(uv)$ for a pipe uv in an instance φ^*, we arrange maximal blocks of consecutive stable edges into a *bundle* that takes a single position in the order, and store the order among the stable edges in the bundle in a separate linked list. We can substitute each bundle with one *representative* stable component. Then, using $\pi_{\varphi^{(2)}}(\langle uv\rangle w)$ for all pipes $\langle uv\rangle w$ incident to $\langle uv\rangle$ we can obtain a list of at most $\deg(\langle uv\rangle) + c$ representative stable components, where c is the number of components of $G^{\diamond}_{\langle uv\rangle}$. We can proceed by embedding all components in $G^{\diamond}_{\langle uv\rangle}$ and the representatives of the remaining stable components in $D_{\langle uv\rangle}$. Obtaining a Jordan curve that encloses all vertices in $(\varphi^{(1)})^{-1}[u]$ now takes $O(E(G^{\diamond}_{\langle uv\rangle})+\deg(\langle uv\rangle))$ time. The order in which the Jordan curve crosses the edges in $G_{\langle uv\rangle}$ defines $\pi_{\varphi^{(1)}}(uv)$, where the size of $\pi_{\varphi^{(1)}}(uv)$ is $O(E(G^{\diamond}_{\langle uv\rangle})+\deg(\langle uv\rangle))$ and each pipe-edge obtained from a representative stable component represents a bundle of stable edges. We can merge consecutive bundles of stable edges in $\pi_{\varphi^{(1)}}(uv)$, as needed, in $O(E(G^{\diamond}_{\langle uv\rangle}) + \deg(\langle uv\rangle))$ time. By (B4), this running time is bounded above by the running time of our implementation of $\mathsf{pipeExpansion}(uv)$, as argued in the proof of Lemma 3.4.8. Therefore, we can reverse every operation and obtain an embedding $\psi_\varphi : G \to \mathcal{H}$ in $O(m \log m)$ time. This completes the proof of Theorem 3.1.1(ii).

3.6 Algorithm for nonorientable surfaces

We show that our algorithm can be adapted to recognize weak embeddings $\varphi : G \to H$ when H is embedded in a *nonorientable* manifold M. We discuss the adaptation to nonorientable surfaces in a separate section to reduce notational clutter in Sections 3.2–3.4.

First, we adapt the definition of the strips system. The embedding of a graph H into a (orientable or nonorientable) surface M is given by a rotation system that specifies, for each vertex of H, the ccw cyclic order of incident edges, and a signature $\lambda : E(H) \to \{-1, 1\}$. To define the strip system $\mathcal{H}$, we proceed exactly as in the case that M is orientable (Section 3.2), except that for every edge $e = uv$, if $\lambda(e) = -1$, we identify $A_{u,v}$ with ∂R_{uv} via an orientation reversing homeomorphism and $A_{v,u}$ with ∂R_{uv} via an orientation preserving homeomorphism, or vice-versa. If we represent M as a sphere with a finite number holes, that are turned into *cross-caps*, the signature of an edge is interpreted as the parity of the number of times an edge passes through a cross-cap.

Second, we adapt the operation of cluster expansion as follows. We put $\lambda(u_v v) := \lambda(uv)$ for all neighbors v of u, and $\lambda(e) := 1$ for all newly created edges e.

Third, we adapt the operation $\mathsf{pipeExpansion}(uv)$ as follows. If $\lambda(uv) = -1$, before creating the cluster $\langle uv \rangle$, we flip the value of λ from -1 to 1, and vice-versa, for every edge of H adjacent to u. This corresponds to pushing the edge uv off all the cross-caps that it passes through. Then the values of λ on the edges incident to $\langle uv \rangle$ in H^* are naturally inherited from the values of λ on the edges adjacent to u and v in H. The value of λ for all other edges in H^* remain the same as in H.

The first two phases of the algorithm remain the same except that they use the adapted operations of cluster and pipe expansion. Phase 3 is modified as follows. If a thick cycle C in G' satisfies $\Pi_{e \in E(\varphi'(C))} \lambda(e) = 1$ (that is, the strip system of $\varphi'(C)$ is homeomorphic to an annulus), we proceed as in the orientable case. The modifications affect only the thick cycles C such that $\Pi_{e \in E(\varphi'(C))} \lambda(e) = -1$, or in other words thick cycles C, for which the strip system of $\varphi'(C)$ is homeomorphic to

the Möbius band. For such a thick cycle C, we report that the instance is negative if C winds more than two times around $\varphi'(C)$; or if all pipe edges of C are thick edges, the underlying graph of C is planar (resp., not planar), and C winds exactly once (resp., twice) around $\varphi'(C)$. Furthermore, we report that the instance is negative if there exist two distinct thick cycles C_1 and C_2 in G' winding once around $\varphi'(C)$ such that $\varphi'(C) = \varphi'(C_1) = \varphi'(C_2)$ in H'. Else we can report that φ is a weak embedding at the end.

The correctness of the algorithm in the Möbius band case is implied by a stronger statement in [FK18, Section 8], but we can easily verify it by the following arguments. Suppose two or more cycles wind exactly once around $\varphi'(C)$. In any embedding φ, the total orders in π_φ (defined in Section 3.2.3) reveres in each traversal of the cycle, which is a contradiction. Similarly, if C winds $k > 2$ many times around $\varphi'(C)$ we divide it into k paths $P_1, \ldots, P_k$ sharing end vertices each of which is mapped injectively into $\varphi'(C)$. We assume that in π_φ the paths $P_1, \ldots, P_k$ appear in the given order up to the choice of orientation. The order of P_i's in π_φ reverses with respect to a fixed orientation if we traverse $\varphi'(C)$. Hence, P_1 must precede and follow P_k along C and therefore $k \leq 2$, a contradiction.

Chapter 4

NP-hardness for Higher Dimensions

This chapter discusses weak embeddings of simplicial 2-complexes. In particular, we show that recognizing weak embeddings of simplicial k-complexes in $\mathbb{R}^3$ for $k = 2$ is NP-hard. Notice that de Mesmay et al. [dMRST18] recently proved that deciding whether a simplicial 2-complex embeds in $\mathbb{R}^3$ is NP-hard. That already implies our claim: given an arbitrary simplicial 2-complex A, let $\varphi : A \to \mathbb{R}^3$ be a function that maps every point in A to $(0, 0, 0)$. Then φ is a weak embedding if and only if A can be embedded in $\mathbb{R}^3$. Our proof is much simpler than [dMRST18], giving a simpler intuition on why recognizing weak embeddings is hard, and also implies stronger results in the weak embedding setting, proving hardness for special cases where the complex is homeomorphic to a disk and at most 9 triangles overlap at any point.

We reduce the problem from Flat-Foldability of origami crease patterns. Flat-Foldability of general crease patterns was first claimed to be hard over twenty years ago [BH96]. We point out a mistake in the hardness reduction in [BH96] and prove that the problem remains NP-complete even for a strict subset of inputs called *box pleating*, defined below. In addition, we provide new terminology to implicitly represent the global layer order of a flat folding, and present a new planar reduction framework from NAE-3SAT for grid-aligned gadgets. The results in

this chapter are joint work with Kenneth C. Cheung, Erik D. Demaine, Takashi Horiyama, Thomas C. Hull, Jason S. Ku, Tomohiro Tachi, and Ryuhei Uehara published in [ACD+16].

4.1 Introduction

In their seminal 1996 paper, Bern and Hayes initiated investigation into the computational complexity of origami [BH96]. They claimed that it is NP-hard to determine whether a given general crease pattern can be folded flat (Flat-Foldability), both when the creases have and have not been assigned crease directions (mountain fold or valley fold). Since that time, there has been considerable work in analyzing the computational complexity of other origami related problems. For example, Arkin et al. [ABD+04] proved that Flat-Foldability is weakly NP-hard even for simple folds, Akitaya et al. [ADK17] strengthened this result to strong NP-hardness, and Demaine et al. [DFL10] proved that optimal circle packing for origami design is also NP-hard.

While the gadgets in the hardness proof presented in [BH96] for unassigned crease patterns are relatively straightforward, their gadgets for assigned crease patterns are considerably more convoluted, and quite difficult to check. In fact, we have found an error in their unassigned crossover gadget where signals are not guaranteed to transmit correctly for wires that do not cross orthogonally, which is required in their construction. Part of the reason no one found this error until now is that there was no formal framework in which to prove statements about flat-folded states. We attempt to provide such a framework.

At the end of their paper, Bern and Hayes pose some interesting open questions to further their work. While most of them have been investigated since, two in particular (problems 2 and 3) have remained untouched until now. First, is there a simpler way to achieve a proof for assigned crease patterns (i.e. "without tabs")? Second, is deciding Flat-Foldability easy under more restrictive inputs? The reductions in [BH96] construct creases at a variety of unconstrained angles. *Box*

pleating is an example of a more restrictive set of inputs. It involves folding creases only along on a subset of a square grid and the diagonals of the squares, a special case of particular interest in transformational robotics and self-assembly [HAB+10], with a universality result constructing arbitrary polycubes using box pleating [BDDO10].

In this chapter we address both of these questions. We prove that FLAT-FOLDABILITY on box-pleated crease patterns inputs is NP-complete in both the unassigned and assigned cases, using relatively simple gadgets containing no more than 25 layers at any point. We then show that this result implies NP-completeness of deciding whether a map from a simplicial 2-complex to $\mathbb{R}^3$ is a weak embedding. The problems remain hard even for inputs $\varphi : A \to \mathbb{R}^3$ such that A is homeomorphic to a disk, $\varphi(A)$ is contained on a single plane, and $|\varphi^{-1}[p]| \leq 9$ for every $p \in \mathbb{R}^3$, i.e., φ maps at most 9 distinct points of A to an image point in $\mathbb{R}^3$.

4.2 Definitions

In general, we are guided by the terminology laid out in [DO07] and [Rob77]. In this chapter, a paper P is a simple polygon in $\mathbb{R}^2$. An *isometric flat folding* of a paper P is a function $f : P \to \mathbb{R}^2$ such that if γ is a piecewise-geodesic curve on P parameterized with respect to arc-length, then $f(\gamma)$ is also a piecewise-geodesic curve parameterized with respect to arc-length. It is not hard to show that under these conditions f must be continuous and non-expansive. Let X_f be the boundary of a paper P together with the set of points not differentiable under f. Then one can prove that X_f is a straight-line graph embedded in P [Rob77], with vertex set V_f and edge set C_f, the *creases* of our folding f. A vertex (resp., crease) in V_f (resp., C_f) is *external* if it contains a boundary point of P, and *internal* otherwise. Subtracting X_f from P results in a disconnected set of open polygons F_f we call *faces*. For any face $F \in F_f$, $f(F)$ is either an isotopic transformation in $\mathbb{R}^2$, or the transformation involves a reflection and is anisotopic. Define $u_f : P \setminus X_f \to \{-1, 1\}$ such that $u_f(p) = -1$ if the face containing p is reflected under f and $u_f(p) = 1$ otherwise. We call $u_f(p)$ the *orientation* of the face containing p. Every point in

P is in exactly one of V_f, C_f, or F_f. We call this partition of P the *isometrically flat foldable crease pattern* $\Sigma_f = (V_f, C_f, F_f)$ induced by f. We call a folding *box pleating* if every vertex lies on the two-dimensional integer lattice, and the creases are aligned at multiples of 45° to each other.

We say two disjoint simply connected subsets of P are *adjacent* to each other if their closures intersect; we call such an intersection the *adjacency* of the adjacent subsets. We say a simply connected subset of P is *uncreased* under f if f is injective when restricted to the subset. We say two simply connected subsets of P *overlap* under f if the interiors of their images under f intersect. We say two simply connected subsets of P *strictly overlap* under f if their images under f exactly coincide. It is known that the set of creases adjacent to an internal vertex of a crease pattern obey the so-called Kawasaki-Justin Theorem: the alternating sum of angles between consecutive creases when cyclically ordered around the vertex equals zero [DO07]. This condition turns out to be necessary and sufficient: given a paper P exhaustively partitioned into a set of isolated points V, open line segments C, and open topological disks F such that every point in V is adjacent to more than two segments in C, then (V, C, F) is an isometrically flat foldable crease pattern induced by a unique isometric flat folding if and only if (V, C, F) obeys the Kawasaki-Justin Theorem.

Let a function $\lambda_f : P \times P \to \{-1, 1\}$ be a *global layer ordering* of an isometric flat folding f if it obeys the following six properties.

Existence: λ_f satisfies *existence* if $\lambda_f(p, q)$ is defined for every distinct pair of points p and q that strictly overlap under f and at least one of p or q is not in X_f; otherwise $\lambda_f(p, q)$ is undefined. Informally, order is only defined between a point on a face and another point overlapping it in the folding.

Antisymmetry: λ_f is *antisymmetric* if $\lambda_f(p, q) = -\lambda_f(q, p)$, where λ_f is defined. Informally, if p is above q, then q is below p.

Transitivity: λ_f is *transitive* if $\lambda_f(p, q) = \lambda_f(q, r)$ implies $\lambda_f(p, r) = \lambda_f(p, q)$, where λ_f is defined. Informally, if q is above p and r is above q, then r is above p.

Consistency (Tortilla-Tortilla Property): For any two uncreased simply

connected subsets O_1 and O_2 of P that strictly overlap under f, λ_f is *consistent* if $\lambda_f(p_1, p_2)$ has the same value for all $(p_1, p_2) \in O_1 \times O_2$, where λ_f is defined. See Figure 4.1. Informally, if two regions completely overlap in the folding, one must be entirely above the other.

Face-Crease Non-Crossing (Taco-Tortilla Property): For any three uncreased simply connected subsets O_1, O_2, and O_3 of P such that O_1 and O_3 are adjacent and strictly overlap, and O_2 overlaps the adjacency between O_1 and O_3 under f, λ_f is *face-crease non-crossing* if $\lambda_f(p_1, p_2) = -\lambda_f(p_2, p_3)$ for any points $(p_1, p_2, p_3) \in O_1 \times O_2 \times O_3$, where λ_f is defined. See Figure 4.1. Informally, if a region overlaps a nonadjacent internal crease, the region cannot be between the regions adjacent to the crease.

Crease-Crease Non-crossing (Taco-Taco Property): For any two adjacent pairs of uncreased simply connected subsets (O_1, O_2) and (O_3, O_4) of P such that every pair of subsets strictly overlap and the adjacency of O_1 and O_2 strictly overlaps the adjacency of O_3 and O_4 under f, λ_f is *crease-crease non-crossing* if either $\{\lambda_f(p_1, p_3), \lambda_f(p_1, p_4), \lambda_f(p_2, p_3), \lambda_f(p_2, p_4)\}$ are all the same or half are $+1$ and half are -1, for any point $(p_1, p_2, p_3, p_4) \in O_1 \times O_2 \times O_3 \times O_4$, where λ_f is defined. See Figure 4.2. Informally, if two creases overlap in the folding, either the regions incident to one crease lie entirely above the regions incident to the other (all same), or the regions incident to one crease nest inside the regions incident to the other (half-half).

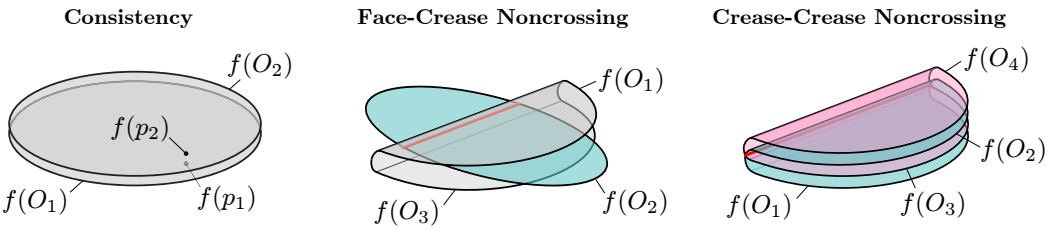


Figure 4.1: Topologically different local interactions within an isometric flat folding. Forbidden configurations are shown for Face-Crease and Crease-Crease Non-Crossing. Crossings are shown with a red segment.

If there exists a global layer ordering for a given isometrically flat foldable

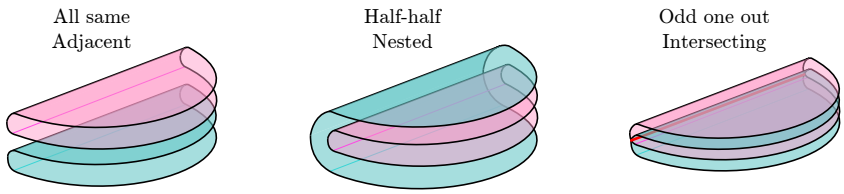


Figure 4.2: Local interaction between overlapping regions around two distinct creases.

crease pattern, we say the crease pattern is *globally flat foldable*. Consider an isometrically flat foldable crease pattern Σ_f containing two adjacent uncreased simply connected subsets O_1 and O_2 of P that strictly overlap under f, and let p and q be points in O_1 and O_2 respectively that overlap under f. Then, O_1 and O_2 are subsets of disjoint adjacent faces of the crease pattern mutually bounding a crease. If λ_f is a global flat folding of Σ_f, then it induces a *mountain/valley assignment* $\alpha_{\lambda_f}(c) = u(p)\lambda_f(p,q)$ for each crease point c in the adjacency of O_1 and O_2. This assignment is unique by consistency. We call a crease point c a *valley fold* (V) if $\alpha_{\lambda_f}(c) = 1$ and a *mountain fold* (M) if $\alpha_{\lambda_f}(c) = -1$. In the figures, mountain folds are drawn in red while valley folds are drawn in blue. By convention, if $\lambda_f(p,q) = -1$ we say that p is *above* q, and if $\lambda_f(p,q) = 1$ we say that p is *below* q.

Given an isometrically flat foldable crease pattern Σ_f, the UNASSIGNED-FLAT-FOLDABILITY problem asks whether there exists a global layer ordering for f. Alternatively, given an isometrically flat foldable crease pattern Σ_f and an assignment $\alpha : C_f \to \{M, V\}$ labeling creases as either mountain or valley, the ASSIGNED-FLAT-FOLDABILITY problem asks whether there exists a global layer ordering for f whose induced mountain valley assignment is consistent with α.

We now prove the following implied properties of globally flat foldable crease patterns relating the layer order between points contained in multiple overlapping faces. Informally, *Pleat-Consistency* says if a face is adjacent and overlapping two larger faces, then the creases between them must have different M/V assignment, forming a pleat. *Path-Consistency* says that a face overlapping creases connecting an adjacent sequence of faces is either above or below all of them.

Lemma 4.2.1. (Pleat-Consistency) *If Σ_f is a globally flat foldable crease pattern containing disjoint uncreased simply connected subsets O_1, O_2, and O_3 of P with O_2 adjacent to both O_1 and O_3 such that O_2 strictly overlaps subsets $O_1' \subset O_1$ and $O_3' \subset O_3$, and the interiors of O_1 and O_3 overlap the adjacencies of O_2, O_3 and O_1, O_2 respectively, then $\lambda_f(p_1, p_2) = \lambda_f(p_2, p_3)$ for any pairwise overlapping points $(p_1, p_2, p_3) \in O_1 \times O_2 \times O_3$.*

Proof. Taco-Tortilla applied to O_3 which overlaps the adjacency of strictly overlapping sets O_2 and O_1' implies $\lambda_f(p_2, p_3) = -\lambda_f(p_3, p_1)$. Similarly, Taco-Tortilla applied to O_1 which overlaps the adjacency of strictly overlapping sets O_3' and O_2 implies $\lambda_f(p_3, p_1) = -\lambda_f(p_1, p_2)$, so $\lambda_f(p_1, p_2) = \lambda_f(p_2, p_3)$. □

Lemma 4.2.2. (Path-Consistency) *If Σ_f is a globally flat foldable crease pattern containing uncreased simply connected subset T of P and a disjoint sequence of adjacent uncreased simply connected subsets $O_1, \ldots, O_n$ of P such that O_i strictly overlaps some subset T_i of T and the interior of O overlaps the adjacency of each pair O_i and O_{i+1} for $i = \{1, \ldots, n-1\}$, then $\lambda_f(t_j, p_j) = \lambda_f(t_k, p_k)$ for any two pairs of overlapping points $(t_j, p_j) \in T_j \times O_j$ and $(t_k, p_k) \in T_k \times O_k$ for $j, k \in \{1, \ldots, n\}$.*

Proof. If some O_i and O_{i+1} overlap, Taco-Tortilla and Consistency ensure that both are at the same side of T, i.e., $\lambda_f(t_i, p_i) = \lambda_f(t_{i+1}, p_{i+1})$ for $(t_i, p_i) \in T_i \times O_i$ and $(t_{i+1}, p_{i+1}) \in T_{i+1} \times O_{i+1}$. Alternatively, O_i and O_{i+1} do not overlap and the closure of $O_i \cup O_{i+1}$ is an uncreased region for which $\lambda_f(t_i, p_i) = \lambda_f(t_{i+1}, p_{i+1})$ by consistency. Applying sequentially to each pair of faces proves the claim. □

The proofs in Section 4.5 and 4.6 contain many examples of the application of these properties. When proving the existence of a global layer ordering λ_f, it is often impractical to define λ_f between every pair of points. Frequently λ_f is uniquely induced by a M/V assignment, consistency, and transitivity. When it is not, we will provide λ_f between additional point pairs so that it will be. We present crease patterns with this implicit layer ordering information and encourage readers to fold them to reconstruct the unique layer orderings they induce.

4.3 Bern and Hayes and k-Layer-Flat-Foldability

Two crossover gadgets are presented in the reduction to Unassigned-Flat-Foldability provided in [BH96]. One is identical to the one used in this chapter (Figure 4.5(b)) and the other is shown in Figure 4.3. They consist of two intersecting pleats. For each, they claim that the M/V assignment of the crease pair intersecting one edge of the gadget deterministically implies the M/V assignment of the crease pair on the opposite side. This claim is true for their perpendicular crossover gadget, but is unfortunately not true for the other for wires meeting at 45°, see Figure 4.3. The gadget as described in [BH96] requires an exterior 45° angle between incoming wires that is the smallest angle at a four-crease vertex, forbidding the wires to be independently assigned by Pleat-Consistency. For completeness, we have also checked the family of possible gadgets of this form, with a rotated internal parallelogram, and no choice of rotation allows the gadget to function correctly as a crossover for the range of widths of wires that appear in the construction. Our proof to follow only uses the perpendicular crossover, avoiding this complication.

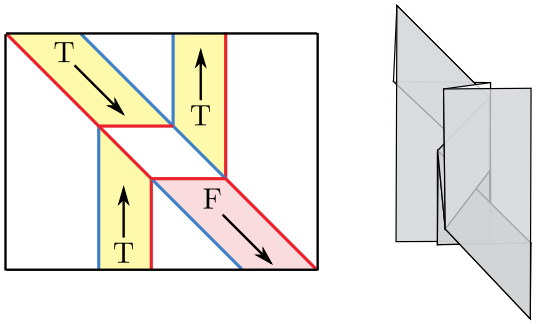


Figure 4.3: Error in the crossover gadget in [BH96].

Also in [BH96], Bern and Hayes define k-Layer-Flat-Foldability to be the same as Unassigned-Flat-Foldability or Assigned-Flat-Foldability but with the additional constraint that f maps at most k distinct points to the same point. They claim that their reduction implies hardness of Unassigned-k-Layer-Flat-Foldability for $k = 7$. But in fact their perpendicular crossover gadget requires nine points to be mapped to the same point. Our reduction uses the same gadget as a crossover, so we reconfirm that Unassigned-k-Layer-Flat-

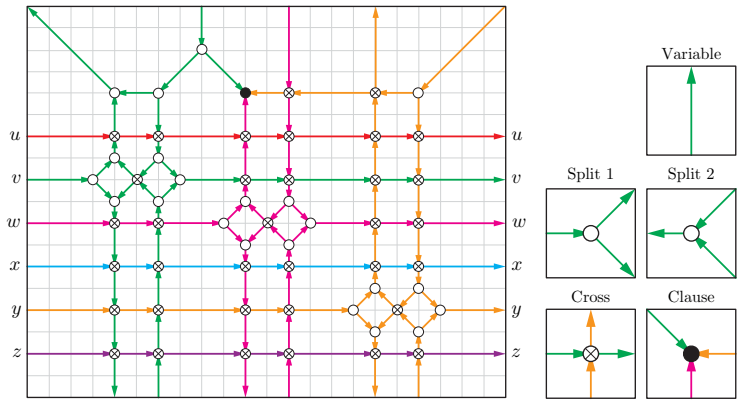


Figure 4.4: SCN Gadgets. [Left] A Complex Clause Gadget constructed from the Not-All-Equal clause on variables v, w, and y of a NAE3-SAT instance on six variables. [Right] The five elemental SCN Gadgets.

Foldability is NP-complete for $k = 9$, even for box pleated crease patterns. Also, because of the complexity of their assigned crease pattern reduction, they were unable to bound the number of layers in their reduction. We explicitly provide gadgets for the assigned case to prove Assigned-k-Layer-Flat-Foldability is NP-complete for $k = 25$, even for box pleated crease patterns.

4.4 SCN-Satisfiability

Our reductions will be from the following NP-complete problem [Sch78].

Definition (Not-All-Equal 3-SAT) Given a collection of clauses each containing three variables, Not-All-Equal 3-SAT (NAE3-SAT)[1] asks if variables can be assigned True or False so that no clause contains variables of only one assignment.

We can construct a planar directed graph G embedded in $\mathbb{R}^2$ from an instance $\mathcal{N}$ of NAE3-SAT. For each clause, construct a Complex Clause Gadget as the one shown in Figure 4.4. The motivation behind the Complex Clause Gadget is to encode the bipartite graph implicit in $\mathcal{N}$ in a planar grid embedding that can be modularly connected. Each directed edge of the Complex Clause Gadget is associated with a unique variable, and we associate a unique color with each variable. Some variables

[1] This problem is sometimes called 'positive' as variables cannot appear negated within clauses, however we follow the naming convention from [Sch78].

do not participate in the clause and simply form a straight chain of directed segments from left to right. However, the three variables participating in the clause are rerouted to intersect at the black dot. We construct a Complex Clause Gadget for each clause in the instance of NAE3-SAT and chain them together side by side, so the arrows exiting the right side of one enter the left side of another. Graph G has vertices that are adjacent to edges associated with exactly one, two, or three variables. We call these vertices *split*, *cross*, and *clause* vertices respectively. In the figures, they are labeled with white circles, crossed circles, and black circles respectively. We call such a directed graph G a *Split-Cross-Not-All-Equal* (SCN) graph.

Definition (SCN-Satisfiability) Given a SCN graph G, SCN-Satisfiability asks if variables can be assigned True or False so that no clause vertex is adjacent to edges associated with variables of only one assignment.

We introduce SCN-Satisfiability as a useful intermediate problem because it is equivalent to NAE3-SAT but its embedding is planar, lies on a grid, and is constructed only by a small number of local elements. SCN-Satisfiability is equivalent to NAE3-SAT because the bipartite graph connecting SCN variables to clause vertices is exactly the bipartite graph representing $\mathcal{N}$ by construction. However, G has useful structure for many problems. It is planar, the embedding contains edges with only four slopes, and the edges are directed meaning that a variable can be represented locally with respect to that direction. Further G is constructed from only a small number of local elements: a variable gadget, two split gadgets, a cross gadget, and a (simple) clause gadget as shown in Figure 4.4. We call these the five *elemental* SCN Gadgets. If we can simulate each of these gadgets in another context, proving that edges of the same color in each gadget must all have the same value, and edges adjacent to a clause vertex do not all have equal value, we can prove other problems NP-hard. This will be our strategy in the following sections.

Theorem 4.4.1. *If a problem X can simulate the elemental SCN gadgets such that edges of the same color in each gadget have the same value and edges adjacent to a clause vertex do not all have equal value and if the correspondent gadgets in X can be connected consistently, then X is NP-hard.*

4.5 Unassigned Crease Patterns

In this section we present gadgets simulating the elemental SCN gadgets with unassigned crease patterns. They are shown in Figure 4.5.

We define a variable gadget to be a pair of parallel creases (or *pleat*) placed close together having an direction as shown in Figure 4.5(a). By pleat-consistency and transitivity, $\lambda_f(a,b) = \lambda_f(b,c) = \lambda_f(a,c)$, so, local to the gadget, it has exactly two globally flat foldable states. We say the variable is *True* if the face to the right of the variable direction is above the face to left ($\lambda_f(a,c) = 1$), and *False* otherwise. The cross gadget consists of two pleats crossing perpendicularly, see Figure 4.5(b). The split and clause gadgets represent the degree 3 vertices of the SCN graph as shown in Figure 4.5(c) and (d).

Lemma 4.5.1. *The unassigned cross gadget is a globally flat foldable crease pattern if and only if opposite variables are equal.*

Proof. Refer to Figure 4.5(b). Assume global flat foldability. Let A, B, C, D, E, F be the maximal subsets of the faces respectively containing points a, b, c, d, e, f such that every pair strictly overlap. First assume that $\lambda_f(a,b) = \lambda_f(c,d)$. By Taco-Taco with respect to adjacencies A, C and B, D, $\lambda_f(a,d) = \lambda_f(c,b)$. By Taco-Taco with respect to adjacencies A, B and C, D, $\lambda_f(a,c) = -\lambda_f(b,d)$. By Pleat-Consistency on A, C, E, $\lambda_f(a,c) = \lambda_f(c,e)$. By Pleat-Consistency on B, D, F, $\lambda_f(b,d) = \lambda_f(d,f)$. So $\lambda_f(c,e) = -\lambda_f(d,f)$. By Taco-Taco with respect to adjacencies C, D and E, F, $\lambda_f(c,f) = -\lambda_f(d,e)$. By Taco-Taco with respect to adjacencies C, E and D, F, $\lambda_f(c,d) = \lambda_f(e,f)$. Thus because $\lambda_f(a,b) = \lambda_f(e,f)$, the variable on the left has the same value as the one on the right. Alternatively if $\lambda_f(a,b) = -\lambda_f(c,d)$, the same series of arguments yields that $\lambda_f(c,d) = -\lambda_f(e,f)$, so $\lambda_f(a,b) = \lambda_f(e,f)$. Thus if

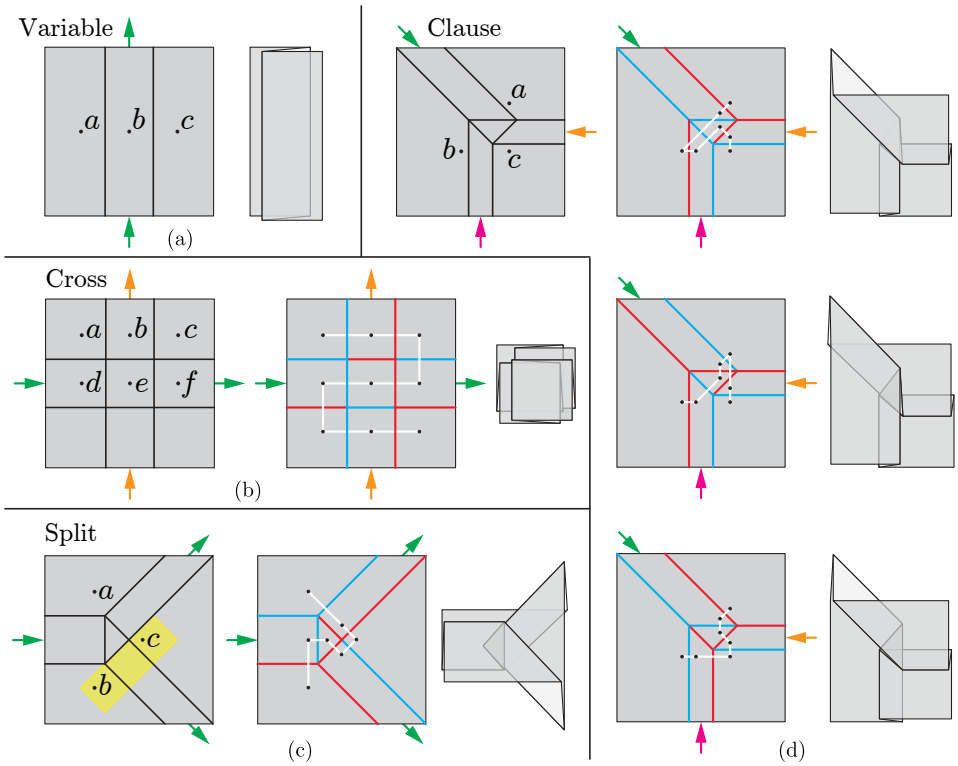


Figure 4.5: Elemental SCN Gadgets simulated with unassigned crease patterns.

global flat foldability holds, opposite variables are equal. Now assume that opposite variables are equal. The M/V assignment in Figure 4.5(b) completely induces λ_f, along with consistency and transitivity. The path shown in Figure 4.5(b) in white is a linear order on the faces satisfying global layer ordering. Further, every other assignment of variables can be represented by a reflection of this crease pattern. □

Lemma 4.5.2. *The unassigned split gadget is a globally flat foldable crease pattern if and only if its three variables are equal.*

Proof. Refer to Figure 4.5(c). Assume global flat foldability. Let A and B be the faces containing points a and b respectively. The region highlighted in the figure and A must satisfy Path-Consistency, so $\lambda_f(a,b) = \lambda_f(a,c)$. Since the crease pattern is symmetric, $\lambda_f(b,a) = \lambda_f(b,c)$. Then, by antisymmetry, $\lambda_f(a,b) = \lambda_f(c,b)$, and therefore all variables are equal. Now assume all variables are equal. The path shown in Figure 4.5(c) is a linear order on the faces satisfying global layer ordering. Further, every other assignment of variables can be represented by a reflection of

this crease pattern. □

Lemma 4.5.3. *The clause gadget is a globally flat foldable crease pattern if and only if its three variables are not all equal.*

Proof. Refer to Figure 4.5(d). Assume for contradiction the clause gadget is globally flat foldable and all variables are equal. By consistency $\lambda_f(a,b) = \lambda_f(b,c) = \lambda_f(c,a)$. By transitivity, $\lambda_f(a,b) = \lambda_f(a,c)$. By antisymmetry, $\lambda_f(a,b) = -\lambda_f(c,a)$, a contradiction. Thus the variables are not all equal. Now assume all variables are not all equal. The paths shown in Figure 4.5(d) are linear orders on the faces satisfying global layer ordering. Further, every other assignment of variables can be represented by the negation of one of these (M/V) assignments. □

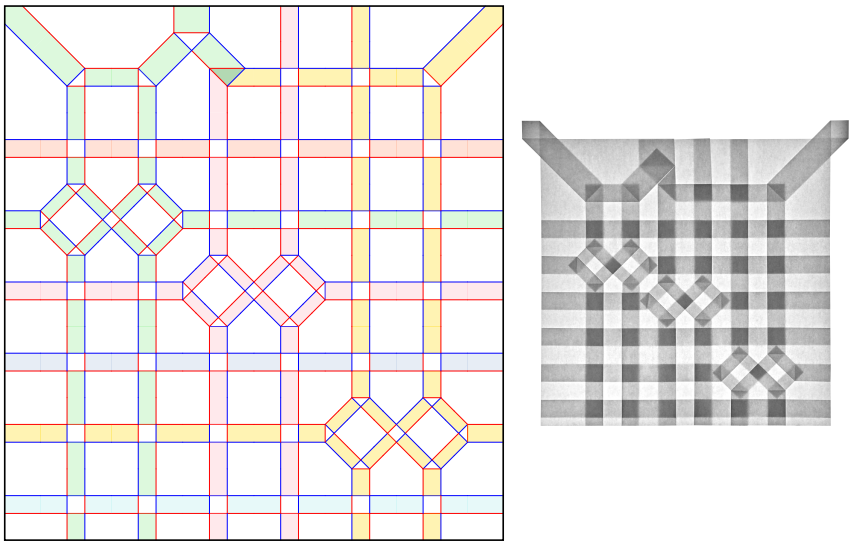


Figure 4.6: A folded example of our unassigned reduction with two clauses on four variables.

Theorem 4.5.4. Unassigned-Flat-Foldability *is NP-complete, even for box pleated crease patterns.*

Proof. Given λ_f as our certificate, we can check in polynomial time whether it satisfies all conditions for global flat foldability, therefore Unassigned-Flat-Foldability is in NP. By Lemma 4.5.1, Lemma 4.5.2, and Lemma 4.5.3, Unassigned-Flat-Foldability can simulate the SCN-Satisfiability gadgets. It remains to check

if the gadgets can be consistently connected. Let the width of a variable be the distance between its parallel creases. The cross gadget connects variables of the same width while the clause and split gadgets both connect variables whose ratios differ by a factor of $\sqrt{2}$. Setting the width of one variable in any gadget induces the width of the other variables in the gadget. Fixing the width of one variable in the Complex Clause Gadget (Figure 4.4), a consistent unique width for all other variables is induced, resulting in the same width for each variable intersecting a left or right edge. Therefore, by Theorem 4.4.1, Unassigned-Flat-Foldability is NP-hard. □

4.6 Assigned Crease Patterns

In this section we present gadgets simulating the elemental SCN gadgets with assigned crease patterns. They are shown in Figure 4.7.

We define a variable gadget as a set of parallel creases placed close together having a direction and a crease assignment as shown in Figure 4.7(a). By Taco-Tortilla, $\lambda_f(a,c) = \lambda_f(b,c) = \lambda_f(a,d) = \lambda_f(b,d)$, so, local to the gadget, it has exactly two globally flat foldable states. We say the variable is *True* if the faces to the right of the variable direction are above the faces to left ($\lambda_f(a,c) = 1$), and *False* otherwise. The cross, split, and clause gadgets are shown in Figure 4.7(b), (c), and (d) respectively.

Lemma 4.6.1. *The assigned cross gadget is a globally flat foldable crease pattern if and only if opposite variables are equal.*

Proof. Refer to Figure 4.7(b). Assume global flat foldability. Let A, B, C, D be the maximal subsets of the faces containing points a, b, c, d, respectively, such that every pair strictly overlap. By transitivity on subset of λ_f induced by the M/V assignment shown, $\lambda_f(a,d) = \lambda_f(b,c) = -1$. By Taco-Taco with respect to adjacencies A, C and B, D, $\lambda_f(a,b) = -\lambda_f(c,d)$. Repeating this argument for adjacent rows of faces all the way down implies $\lambda_f(a,b) = -\lambda_f(c,d) = \lambda_f(e,f) = -\lambda_f(g,h) = \lambda_f(i,j)$. Thus, the variable on the top edge of the gadget has the same value as the one

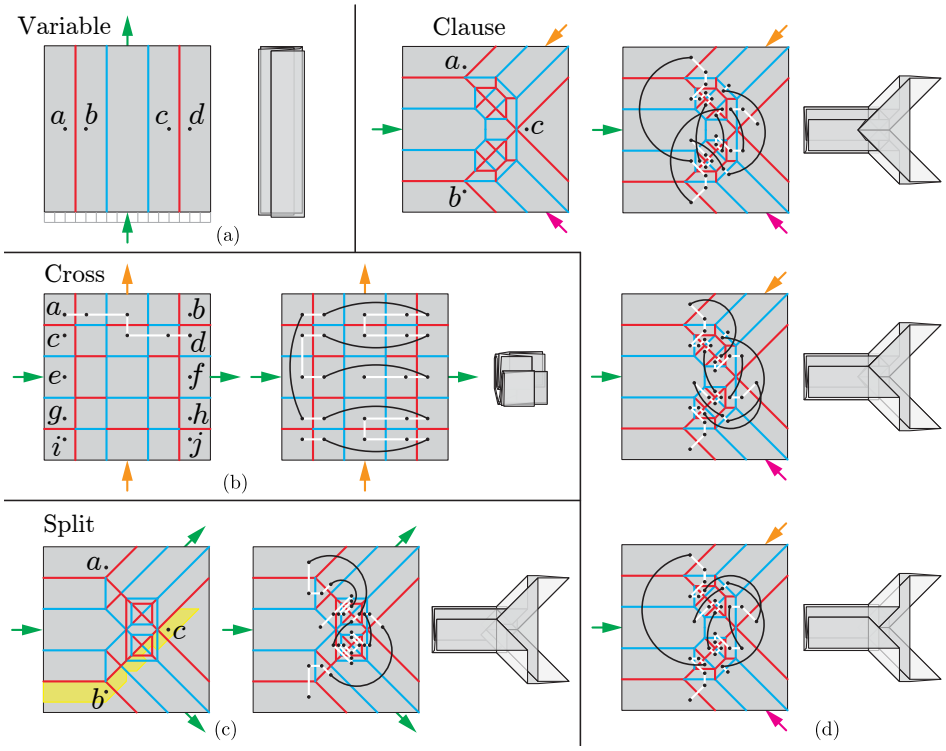


Figure 4.7: Elemental SCN Gadgets simulated with assigned crease patterns.

on the bottom. First assume $\lambda_f(g,a) = \lambda_f(a,b)$. Then previous implications imply $\lambda_f(g,a) = -\lambda_f(g,h)$. By transitivity and antisymmetry, $\lambda_f(g,a) = \lambda_f(h,b)$. Thus, the variable on the left side of the gadget has the same value as the one on the right. Alternatively, assume $-\lambda_f(g,a) = \lambda_f(a,b)$ so $\lambda_f(c,i) = \lambda_f(d,c)$. Then previous implications imply $\lambda_f(c,i) = \lambda_f(i,j)$. By transitivity and antisymmetry, $\lambda_f(c,i) = \lambda_f(d,j)$. Thus, the variable on the left side of the gadget has the same value as the one on the right. So, if globally flat foldable, opposite variables are equal. Now assume that opposite variables are equal. One can fix a unique λ_f by choosing a subset of λ_f in addition to the subset induced by the M/V assignment and consistency. The path shown in the right of Figure 4.7(b) is a linear order on the faces satisfying global layer ordering. White edges represent a relations implied by the M/V assignment, while black edges depend on the choice of λ_f encoding the truth assignment of variables. Further, every other assignment of variables can be represented by a reflection of this crease pattern. □

Lemma 4.6.2. *The assigned split gadget is a globally flat foldable crease pattern if and only if its three variables are equal.*

Proof. Refer to Figure 4.7(c). Assume global flat foldability. Let A and B be the faces containing points a and b respectively. The region highlighted in the figure and A must satisfy Path-Consistency, so $\lambda_f(a,b) = \lambda_f(a,c)$. Since the crease pattern is symmetric, $\lambda_f(b,a) = \lambda_f(b,c)$. Then, by antisymmetry, $\lambda_f(a,b) = \lambda_f(c,b)$, and therefore all variables are equal. Now assume all variables are equal. The path shown in Figure 4.7(c) is a linear order on the faces satisfying global layer ordering. Further, any other assignment of variables can be attained by a reflection. □

Lemma 4.6.3. *The assigned clause gadget is a globally flat foldable crease pattern if and only if its three variables are not all equal.*

Proof. Refer to Figure 4.7(d). Assume for contradiction the clause gadget is global flat foldable and all variables are equal. By consistency $\lambda_f(a,b) = \lambda_f(b,c) = \lambda_f(c,a)$. By transitivity, $\lambda_f(a,b) = \lambda_f(a,c)$. By antisymmetry, $\lambda_f(a,b) = -\lambda_f(c,a)$, a contradiction. Thus the variables are not all equal. Now assume all variables are not all equal. The paths shown in Figure 4.7(d) are linear orders on the faces satisfying global layer ordering. Further, any other assignment of variables can be attained by reversing the arrows in the figure. □

Theorem 4.6.4. Assigned-Flat-Foldability *is NP-complete, even for box pleated crease patterns.*

Proof. Given λ_f as our certificate, we can check in polynomial time whether it satisfies all conditions for global flat foldability and if it is consistent with the crease assignment, therefore Assigned-Flat-Foldability is in NP. By Lemma 4.6.1, Lemma 4.6.2, and Lemma 4.6.3, Assigned-Flat-Foldability can simulate the SCN-Satisfiability gadgets. It remains to check if the gadgets can be consistently connected. Let the width of a variable be the distance between its two parallel mountain creases. By the same argument as in the proof of Theorem 4.5.4, widths

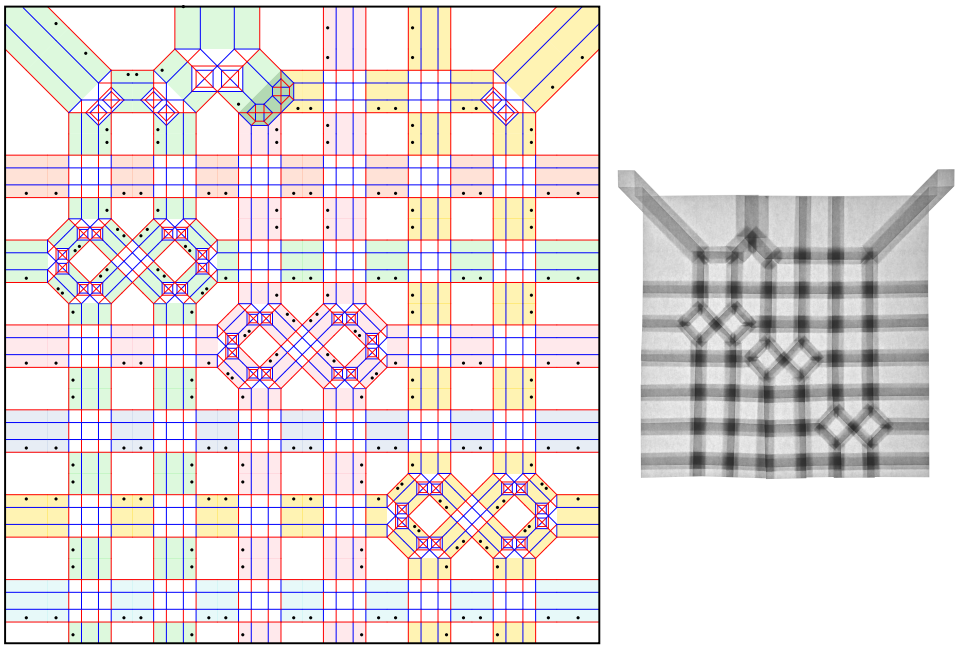


Figure 4.8: A folded example of our assigned reduction with two clauses on four variables. The truth assignment is shown by dots that indicate the layers that fold below.

of variables can be assigned consistently. Therefore, by Theorem 4.4.1, Assigned-Flat-Foldability is NP-hard. □

4.7 Hardness for Weak Embeddings

Given an instance of Flat-Foldability, we can construct a map $\varphi : A \to \mathbb{R}^3$ where A is a simplicial 2-complex as follows. Subdivide each face in F_f at $f^{-1}(X_f)$. Triangulate the obtained faces obtaining the set of triangles T. Every face originates $O(n)$ triangles in T, where n is the number of edges in X_f. Glue every pair of triangles at their common edge. This results in the simplicial 2-complex A. Note that because the paper P of our instance is homeomorphic to a disk, so is A. Let the point $p' \in A$ be the point that corresponds to $p \in P$. Now let φ be such that if $f(p) = (p.x, p.y)$, then $\varphi(p') = (p.x, p.y, 0)$. Given an embedding ψ_ε that approximates φ for some sufficiently small ε, one can obtain λ_f such that the Flat-Foldability constraints are satisfied by the z-order of faces in ψ_ε. Similarly, given an ordering λ_f, the existence of an embedding ψ_ε is implied by Lemma 11.4.1 in [DO07]. We

denote by $\varphi^{-1}[p]$ the set of points in A that are mapped to $p \in \mathbb{R}^3$. In the reduction in the proof of Theorem 4.5.4, the cross gadget has nine faces overlapping at a point. In all other gadgets $|\varphi^{-1}[p]| < 9$ for a point p in the image of the gadget. Then, the following is a consequence of Theorem 4.5.4:

Corollary 4.7.1. *It is NP-hard to decide whether a map $\varphi : A \to \mathbb{R}^3$ is a weak embedding, where A is a simplicial 2-complex, even if A is homeomorphic to a disk, $\varphi(A)$ is contained on a single plane, and $|\varphi^{-1}[p]| \leq 9$ for every $p \in \mathbb{R}^3$.*

Chapter 5

Conclusion

This thesis presented algorithmic results for recognizing weak embeddings of simplicial complexes. We have shown that, in general, the problem is NP-hard (Chapter 4). For simplicial 1-complexes (graphs) in a surface, we showed that the problem can be solved in $O(mn \log mn)$ via an algorithm for graphs with m edges and n vertices; In the special case when the input is a simplicial map, the problem can be solved in $O(m \log m)$ time (Chapter 3). When the input is a (not necessarily simplicial) map from a max-degree-2 graph to the plane, the problem can be solved in $O(n \log n)$ time via an algorithm (Chapter 2). Both algorithms improve on previously known bounds.

A simple lower bound shows that, for max-degree-2 graphs, the upper bound is the best possible (Figure 5.1). It is a reduction from sorting to weak embedding. Given a set of real numbers $\{a_1, \ldots, a_n\}$, let G be the graph containing n copies of

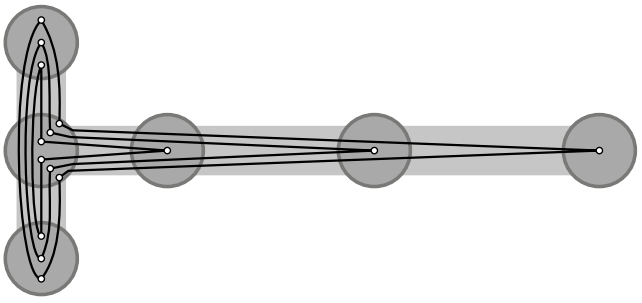


Figure 5.1: Reduction from sorting to weak embedding.

C_5. Set φ so that each cycle of G is mapped to $((0,0),(0,1),(0,-1),(0,0),(a_i,0))$ for each number a_i of the input. Given an embedding ψ_φ that approximates φ, one can recover the sorted order of the elements of the input set in $O(n)$ time since it must correspond to the nesting order of the cycles in ψ_φ. Note that if the cycles are not nested they would intersect near $(0,0)$. Then, the embedding ψ_φ must take $\Omega(n \log n)$ time to be computed.

However this construction does not work for the case when the input graph is connected (equivalent to recognizing a weakly simple polygon). The best known lower bound in this case is $\Omega(n)$, due to reading the whole input (by changing the drawing of any of the edges in an embedding of a graph with more than two edges, one could cause a proper crossing, for example). It is an open problem whether the upper bound can be improved for weakly simple polygons. The $\Omega(n \log n)$ bound is also the best known for general graphs. It is an open problem whether the $O(mn \log mn)$ bound can be improved for recognizing weak embeddings of general graphs.

There are several open problems related to the recognition of weak embeddings. Perhaps the most prominent one is the notoriously difficult cluster-planarity problem (briefly introduced in Section 3.1 of Chapter 3). I hope that the methods explored in this thesis can contribute to a better understanding of this and other related problems.

For instance, our methods could be applied to special cases of this problem as follows. A clustered graph $C = (G, T)$ is given by a graph G and a rooted tree T whose leaves are the vertices of G. A node ν of T that is neither a leaf or the root is called a *cluster* and corresponds to a subset $V(\nu)$ of vertices of G such that $V(\nu_2) \subset V(\nu_1)$ if ν_2 is a descendant of ν_1. A clustered graph $C = (G, T)$ is *flat* if the height of T is 2, i.e., a vertex can only be in a single cluster. A drawing of C maps each cluster ν to a simply connected region $R(\nu)$ that is contained in $R(\nu')$ if ν is a descendant of ν'. The vertices in $V(\nu)$ are drawn in $R(\nu)$ and each edge of G can cross the boundary of a cluster $R(\nu)$ at most once. A clustered graph is *c-planar* if it admits a planar drawing. As stated in Chapter 3, weak embeddings are a special

case of cluster-planar graphs. They correspond to the problem c-planarity with pipes [AL16] which restricts C to be flat and the edges between the same pair of clusters to be "bundled" together in the embedding. If the adjacency graph of a flat clustered graph (considering two clusters to be adjacent if there is an edge between their vertices) is 3-connected, our algorithm can be used to determine c-planarity.

Bibliography

[AAET17] Hugo A. Akitaya, Greg Aloupis, Jeff Erickson, and Csaba D. Tóth. Recognizing weakly simple polygons. *Discrete & Computational Geometry*, 58(4):785–821, 2017.

[ABD+04] 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(1):23–46, 2004.

[ACD+16] Hugo A. Akitaya, Kenneth C. Cheung, Erik D. Demaine, Takashi Horiyama, Thomas C. Hull, Jason S. Ku, Tomohiro Tachi, and Ryuhei Uehara. Box pleating is hard. In Jin Akiyama, Hiro Ito, Toshinori Sakai, and Yushi Uno, editors, *Discrete and Computational Geometry and Graphs*, LNCS, pages 167–179. Springer, Cham, 2016.

[ADK17] Hugo A. Akitaya, Erik D. Demaine, and Jason S. Ku. Simple folding is really hard. *Journal of Information Processing*, 25:580–589, 2017.

[ADLDBF17] Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, and Fabrizio Frati. Strip planarity testing for embedded planar graphs. *Algorithmica*, 77(4):1022–1059, 2017.

[AFT18] Hugo A. Akitaya, Radoslav Fulek, and Csaba D Tóth. Recognizing weak embeddings of graphs. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 274–292. SIAM, 2018.

[AL16] Patrizio Angelini and Giordano Da Lozzo. Clustered planarity with pipes. In Seok-Hee Hong, editor, *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, volume 64 of *LIPIcs*, pages 13:1–13:13. Schloss Dagstuhl, 2016.

[BCD+02] Michael A. Bender, Richard Cole, Erik D. Demaine, Martin Farach-Colton, and Jack Zito. Two simplified algorithms for maintaining order in a list. In *European Symposium on Algorithms*, pages 152–164. Springer, 2002.

[BD09] Henning Bruhn and Reinhard Diestel. MacLane's theorem for arbitrary surfaces. *J. Combin. Theory Ser. B*, 99:275–286, 2009.

[BDDO10] Nadia M. Benbernou, Erik D. Demaine, Martin L. Demaine, and Aviv Ovadya. Universal hinge patterns to fold orthogonal shapes. In *Origami*^{5}, pages 405–420. A K Peters, Singapore, July 2010.

[Bel83] S. B. Belyi. Self-nonintersecting and non intersecting chains. *Mathematical notes of the Academy of Sciences of the USSR*, 34(4):802–804, 1983. Translated from Matematicheskie Zametki, Vol. 34, No. 4, pp. 625–628, 1983.

[BH96] Marshall Bern and Barry Hayes. The complexity of flat origami. In *Proc of the 7th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 175–183, 1996.

[CDBPP09] 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.

[CDR02] 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 geometric objects in* $\mathbb{R}^3$

(Las Vegas, NV, 2001), volume 304 of *Contemp. Math.*, pages 287–311. Amer. Math. Soc., Providence, RI, 2002.

[CEX15] Hsien-Chih Chang, Jeff Erickson, and Chao Xu. Detecting weakly simple polygons. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1655–1670, 2015.

[CH05] Richard Cole and Ramesh Hariharan. Dynamic LCA queries on trees. *SIAM Journal on Computing*, 34(4):894–923, 2005.

[Cha91] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991.

[Cro07] Christoffer Cromwik. *Numerical folding of airbags based on optimization and origami*. PhD thesis, Chalmers University of Technology and Göteborg University, 2007.

[dBvKOS00] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, Heidelberg, 2000.

[DFL10] Erik D. Demaine, Sándor P. Fekete, and Robert J. Lang. Circle packing for origami design is hard. In *Origami*^{5}, pages 609–626. A K Peters, Singapore, July 2010.

[Die17] Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg, fifth edition, 2017.

[dMRST18] Arnaud de Mesmay, Yo'av Rieck, Eric Sedgwick, and Martin Tancer. Embeddability in $\mathbb{R}^3$ is NP-hard. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1316–1329, 2018.

[DO07] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms*. Cambridge University Press, Cambridge, 2007.

[DS87] Paul Dietz and Daniel Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the 19th ACM Symposium on Theory of Computing (STOC)*, pages 365–372. ACM, 1987.

[DT96] Giuseppe Di Battista and Roberto Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25:956–997, 1996.

[EHB04] James T. Early, Roderick Hyde, and Richard L. Baron. Twenty-meter space telescope based on diffractive fresnel lens. In *UV/Optical/IR Space Telescopes: Innovative Technologies and Concepts*, volume 5166, pages 148–157. International Society for Optics and Photonics, 2004.

[FCE95a] Qing-Wen Feng, Robert F. Cohen, and Peter Eades. How to draw a planar clustered graph. In *Proceedings of the 1st Conference on Computing and Combinatorics (COCOON)*, volume 959 of *LNCS*, pages 21–30. Springer, Berlin, 1995.

[FCE95b] Qing-Wen Feng, Robert F. Cohen, and Peter Eades. Planarity for clustered graphs. In *Proceedings of the 3rd European Symposium on Algorithms (ESA)*, volume 979 of *LNCS*, pages 213–226, Berlin, 1995.

[FK18] Radoslav Fulek and Jan Kynčl. Hanani-Tutte for Approximating Maps of Graphs. In Bettina Speckmann and Csaba D. Tóth, editors, *Proceedings of the 34th International Symposium on Computational Geometry (SoCG)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:15, Dagstuhl, Germany, 2018.

[FT17] Andrea Francke and Csaba D. Tóth. A census of plane graphs with polyline edges. *SIAM Journal on Discrete Mathematics*, 31(2):1174–1195, 2017.

[Ful17] Radoslav Fulek. Embedding Graphs into Embedded Graphs. In Yoshio Okamoto and Takeshi Tokuyama, editors, *28th Internatio-*

nal Symposium on Algorithms and Computation (ISAAC 2017), volume 92 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:12, Dagstuhl, Germany, 2017.

[Grü12] Branko Grünbaum. Polygons: Meister was right and poinsot was wrong but prevailed. *Beiträge zur Algebra und Geometrie/Contributions to Algebra and Geometry*, 53(1):57–71, 2012.

[HAB+10] E. Hawkes, B. An, Nadia M. Benbernou, H. Tanaka, S. Kim, E.D. Demaine, D. Rus, and Robert J. Wood. Programmable matter by folding. *Proceedings of the National Academy of Sciences*, 107(28):12441–12445, 2010.

[Han34] Haim Hanani. Über wesentlich unplättbare Kurven im dreidimensionalen Raume. *Fundamenta Mathematicae*, 23:135–142, 1934.

[Hat11] Koshiro Hatori. History of origami in the east and the west before interfusion. In *Origami^5: Fifth International Meeting of Origami Science, Mathematics, and Education*, pages 14–22. CRC Press, 2011.

[HMM00] Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[HT74] John Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the Association for Computing Machineryf*, 21:549–568, 1974.

[JKX+13] Mustapha Jamal, Sachin S. Kadam, Rui Xiao, Faraz Jivan, Tzia-Ming Onn, Rohan Fernandes, Thao D. Nguyen, and David H Gracias. Tissue engineering: Bio-origami hydrogel scaffolds composed of photocrosslinked peg bilayers (adv. healthcare mater. 8/2013). *Advanced Healthcare Materials*, 2(8):1066–1066, 2013.

[JLM98] Michael Jünger, Sebastian Leipert, and Petra Mutzel. Level planarity testing in linear time. In *Proceedings of the 6th Symposium on Graph Drawing (GD)*, volume 1547 of *LNCS*, pages 224–237. Springer, Berlin, 1998.

[KTY+06] Kaori Kuribayashi, Koichi Tsuchiya, Zhong You, Dacian Tomus, Minoru Umemoto, Takahiro Ito, and Masahiro Sasaki. Self-deployable origami stent grafts as a biomedical application of ni-rich tini shape memory alloy foil. *Materials Science and Engineering: A*, 419(1-2):131–137, 2006.

[Lan03] Robert J. Lang. *Origami Design Secrets: Mathematical Methods for an Ancient Art*. CRC Press, 2003.

[LPL+13] Yang Liu, Jungwook Park, Robert J. Lang, Azita Emami-Neyestanak, Sergio Pellegrino, Mark S. Humayun, and Yu-Chong Tai. Parylene origami structure for intraocular implantation. In *Solid-State Sensors, Actuators and Microsystems (TRANSDUCERS & EUROSENSORS XXVII), 2013 Transducers & Eurosensors XXVII: The 17th International Conference on IS - SN - VO*, pages 1549–1552. IEEE, 2013.

[Min97] Piotr Minc. Embedding of simplicial arcs into the plane. *Topology Proceedings*, 22:305–340, 1997.

[Moh99] Bojan Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics*, 12(1):6–26, 1999.

[Mor06] Ares Ribó Mor. *Realization and counting problems for planar structures*. PhD thesis, Freie Universität Berlin, 2006.

[Rob77] S. A. Robertson. Isometric folding of Riemannian manifolds. *Proceedings of the Royal Society of Edinburgh*, 79(3–4):275–284, 1977.

[RS98] Dušan Repovš and Arkadij B. Skopenkov. A deleted product criterion for approximability of maps by embeddings. *Topology Appl.*, 87(1):1–19, 1998.

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th ACM symposium on Theory of computing (STOC)*, pages 216–226. ACM, 1978.

[SH76] Michael Ian Shamos and Dan Hoey. Geometric intersection problems. In *Proceedings of the 17th Symposium on Foundations of Computer Science (FOCS)*, pages 208–215. IEEE, 1976.

[Sie69] Karol Sieklucki. Realization of mappings. *Fundamenta Mathematicae*, 65:325–343, 1969.

[Sko03] Mikhail Skopenkov. On approximability by embeddings of cycles in the plane. *Topology Appl.*, 134(1):1–22, 2003.

[SKSG13] M. Schenk, S. Kerr, A.M. Smyth, and S.D. Guest. Inflatable cylinders for deployable space structures. In *Proceedings of the 1st Conference Transformables*, pages 18–20, 2013.

[Tho89] Carsten Thomassen. The graph genus problem is NP-complete. *Journal of Algorithms*, 10(4):568–576, 1989.

[Tut70] William T. Tutte. Toward a theory of crossing numbers. *Journal of Combinatorial Theory*, 8:45–53, 1970.

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Sympos. on Theory of Computing (STOC)*, pages 887–898. ACM, 2012.