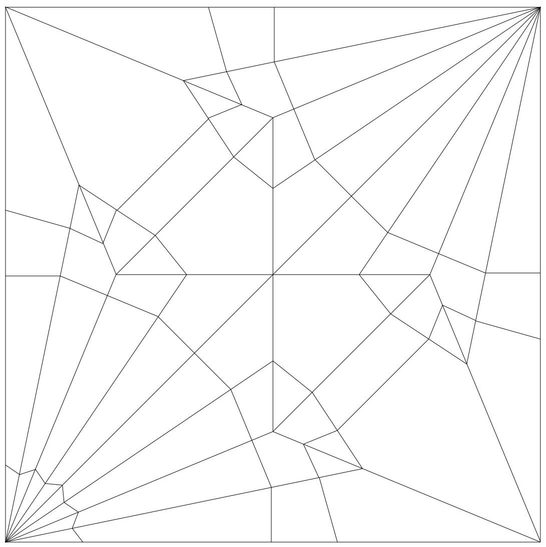


The Mathematics of Origami

Thomas H. Bertschinger, Joseph Slote,
Olivia Claire Spencer, Samuel Vinitsky

Contents

1 Origami Constructions 2

1.1 Axioms of Origami 3

1.2 Lill's method 12

2 General Foldability 14

2.1 Foldings and Knot Theory 17

3 Flat Foldability 20

3.1 Single Vertex Conditions 20

3.2 Multiple Vertex Crease Patterns 23

4 Computational Folding Questions: An Overview 24

4.1 Basics of Algorithmic Analysis 24

4.2 Introduction to Computational Complexity Theory 26

4.3 Computational Complexity of Flat Foldability 27

5 Map Folding: A Computational Problem 28

5.1 Introduction to Maps 28

5.2 Testing the Validity of a Linear Ordering 30

5.3 Complexity of Map Folding 35

6 The Combinatorics of Flat Folding 38

6.1 Definition 39

6.2 Winding Sequences 41

6.3 Enumerating Simple Meanders 48

6.4 Further Study 58

Introduction

Mention of the word "origami" might conjure up images of paper cranes and other representational folded paper forms, a child's pasttime, or an art form. At first thought it would appear there is little to be said about the mathematics of what is by some approximation merely crumpled paper.

Yet there is a surprising amount of conceptual richness to be teased out from between the folds of these paper models. Even though researchers are just at the cusp of understanding the theoretical underpinnings of this ancient art form, many intriguing applications have arisen—in areas as diverse as satellite deployment and internal medicine.

Parallel to the development of these applications, mathematicians have begun to seek descriptions of the capabilities and limitations of origami in a more abstract sense. The continuity of the sheet of paper in combination with the discreteness of a folding sequence lends origami to analysis with numerous mathematical structures and techniques. In many of these areas, initial results have proven exciting.

A natural first mathematical question to about origami is "what models are there? How do we characterize and organize them?" If we conceive of folding as a mapping from crease patterns (the pattern of creases left on a sheet of paper unfolded from a model) to folded models, we might wonder how hard it is, from a computational perspective, to determine whether a crease pattern *can* be folded (quite hard!), and if it can, in how many ways? (it's hard to tell). In fact we will see that the folding ability of origami crease patterns can be characterized with tools from knot theory, potentially leading to a classification of origami models in terms of "folding complexity."

Moreover, from a geometric standpoint, it turns out that if the ancient Greeks had thrown away their compasses and straightedges and merely folded their paper (or papyrus, as it may have been), they would have had much more luck with difficult construction problems. As a one-fold-at-a-time procedure, the geometric-constructive ability of origami is perhaps the best-understood component of the discipline. This is where we will begin.

1 Origami Constructions

Some classical problems dating back to the Ancient Greeks proven unsolvable using a compass and straightedge have been shown to be possible through the more powerful method of origami. However, not all such problems are able to be solved using origami. In order to compare these construction techniques, we must first provide geometric axioms, or operations, for creating folds in the plane.

1.1 Axioms of Origami

To begin, just as Euclid based his studies on a set of fundamental axioms, or postulates, the mathematics of origami is based on a set of axioms as well. There are seven ways single creases can be folded by positioning some combination of lines and points on paper, known as the Huzita-Justin Axioms. The first six axioms were discovered by Jacques Justin in 1989, then rediscovered and reported by Humiaki Huzita in 1991. Axiom 7 was discovered by Koshiro Hatori in 2001, but it turned out all seven axioms were included in Jacques Justin's original paper. These axioms describe the geometric constructions possible through origami. It must be noted that using the term axioms here is misleading, as these axioms are not independent. Each of these axioms can be derived through repeadetly constructing Axiom 6. Though these are deemed "axioms," a term like "operations" may be slightly more appropriate. The first six axioms allow all quadratic, cubic, and quartic equations with rational coefficients to be solved by setting a unit length on a piece of paper then folding to find the roots of polynomials. Using these axioms, it is also possible to construct a regular N-gon for N if N is of the form $2^i3^j(2^k3^l+1)$, such that (2^k3^l+1) is prime [17]. Lastly, two of the three problems of antiquity, the trisection of an angle and the doubling of the cube, can be constructed using these axioms. The seven Huzita-Justin axioms consist of the following:

1. Given two points p_1 and p_2, we can fold a line connecting them.
2. Given two points p_1 and p_2, we can fold p_1 onto p_2.
3. Given two lines l_1 and l_2, we can fold line l_1 onto l_2.
4. Given a point p_1 and a line l_1, we can make a fold perpendicular to l_1 passing through the point p_1.
5. Given two points p_1 and p_2 and a line l_1, we can make a fold that places p_1 onto l_1 and passes through the point p_2.
6. Given two points p_1 and p_2 and two lines l_1 and l_2, we can make a fold that places p_1 onto line l_1 and places p_2 onto line l_2.

7. Given a point p_1 and two lines l_1 and l_2, we can make a fold perpendicular to l_2 that places p_1 onto line l_1. [17]

This seventh axiom does not allow any higher-order equations to be solved than the original six axioms do [17]. However, it is possible to solve higher-order equations through multifolds, or more than one simultaneous crease. The quintisection of an angle, or dividing an angle into five equal parts, requires solving a fifth-order equation, and thus is not solvable using the original seven Huzita-Justin Axioms. However, by using one or more axioms from a set of more complicated axioms defining two simultaneous folds, quintisecting an angle is possible [16]. Thus, theoretically, polynomials of arbitrary degree are solvable if an arbitrary number of simultaneous creases are allowed. This leads us to one of the biggest open problems regarding origami and geometric constructions–which general higher-order polynomials can be translated into folding problems?

The Greek problems of antiquity are a trio of geometric problems whose solutions were attempted only through the use of a compass and straightedge. These problems include angle trisection, cube duplication, and circle squaring. Over 2,000 years after the problems were formulated, they were each proved unsolvable using solely a compass and straightedge. The first problem, angle trisection, can be solved through origami.

Beginning with rectangular or square paper and labeling the corner points A, B, C, and D (beginning at the top left corner and labeling the rest in order counterclockwise), the paper must be creased beginning at point B and meeting line segment $\overline{AD}$, thus denoting the angle θ to be trisected (see fig. 1.2). Next, two horizontal folds must be made parallel to the bottom edge of the paper. The following fold must make both the point where the upper horizontal fold meets the paper's edge to land on the angled line denoting the upper boundary for θ, as well as make point B land on the lower horizontal fold. This fold will cause the lower horizontal fold to be placed at an angle up the page on the folded portion of paper—the next fold will be along this angle, ultimately resulting in a crease running toward point B within the original boundaries of θ. By recreating this fold such that it fully extends to point B, then folding the bottom edge of the paper to meet this fold, a trisected angle θ with three segments extending from point B will have been formed. [2]

Proof. Suppose the three sub-angles of θ are labeled α, β, and γ. Each of these three sections of θ must be equal and must all equal $\frac{\theta}{3}$. $\triangle$EBb has a base $\overline{EB}$ with $\overline{EG} = \overline{GB}$, height $\overline{Gb}$, and sides $\overline{Eb} = \overline{Bb}$, as $\triangle$EBb

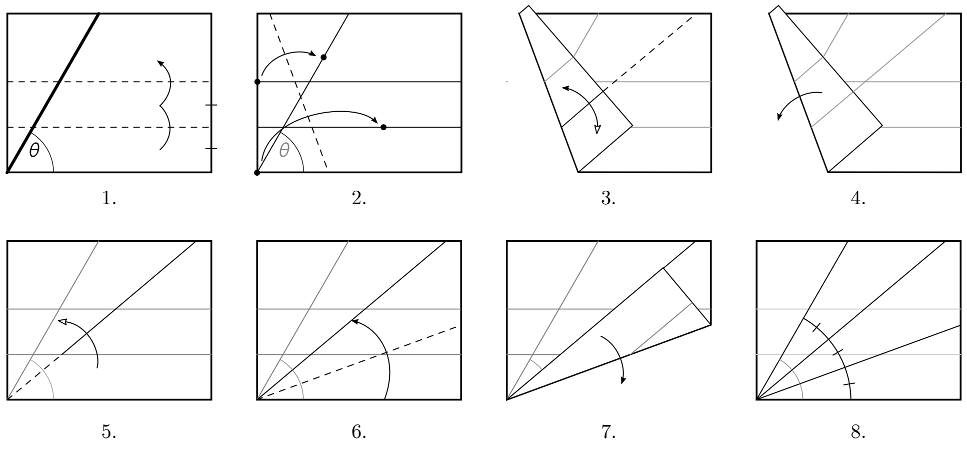


Figure 1.1: Steps to Trisect an Angle

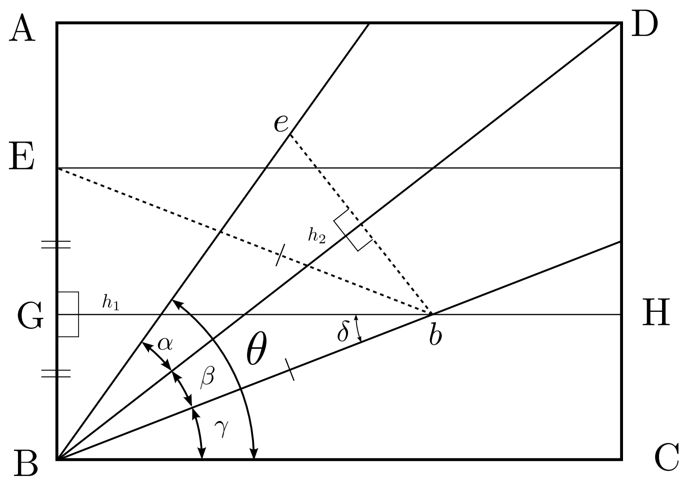


Figure 1.2: Angle Trisection Diagram

is isosceles. The reflection of $\triangle$EBb, $\triangle$ebB, is also isosceles. This height, labeled h_2, implies $\alpha = \beta$. By symmetry, $\beta = \delta$. Since $\overline{GH} \parallel \overline{BC}$, $\gamma = \delta$, and thus $\beta = \gamma$. Therefore, $\alpha = \beta = \gamma = \dfrac{\theta}{3}$. [2] $\square$

The Delian problem (dating back to the civilization of Delos) of doubling the cube is another ancient geometric problem. Given one edge of a cube, the construction of the edge of a second cube whose volume is double that of the first cube is required. In order to construct such a number, here's how we begin. A piece of square paper must first be folded into thirds, then folded such that the bottom right corner of the paper touches the left edge of the page at the same time as the bottom third at S touches the two-thirds line. We will show that where the corner C meets the left edge of the paper is where the ratio of the length of the top of the page to the bottom of the page is $\frac{x}{1} = \sqrt[3]{2}$, or $\frac{\alpha}{\beta} = \sqrt[3]{2}$. Therefore, a cube with side length α will have twice the volume of a cube of side length β. [2]

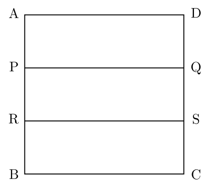


Figure 1.3: Initial Folds

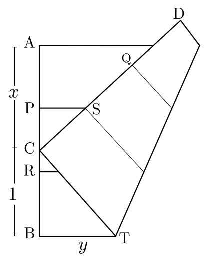


Figure 1.4: Doubling the Cube

Proof. Looking at the diagrams, it follows that point C must meet $\overline{AB}$ while point S simultaneously meets $\overline{PQ}$. We will denote $\overline{BC}$ as the unit length one, $\overline{AC}$ as having length x, and $\overline{BT}$ as having length y. Thus, edge $\overline{AB} = x+1$.

$$\overline{BP} = \frac{2(\overline{AB})}{3} = \frac{2(x+1)}{3}.$$

$$\overline{CP} = \overline{BP}–\overline{CB} = \frac{2(x+1)}{3}–1 = \frac{2x-1}{3}.$$

It follows that $\overline{CT} = x + 1–y$. By the Pythagorean Theorem,

$$\overline{CB}^2 + \overline{BT}^2 = \overline{CT}^2, \text{ so } 1 + y^2 = (x+1-y)^2.$$

Simplification yields $y = \frac{x^2+2x}{2x+2}$. Continuing,

$$\overline{AP} = \frac{\overline{AB}}{3} = \frac{x+1}{3}$$

$$\text{and } \overline{CP} = x - \frac{\overline{AB}}{3} = x - \frac{x+1}{3} = \frac{2x-1}{3}.$$

$$\text{Thus, } \triangle CBT \sim \triangle PSC \Rightarrow \frac{\overline{BT}}{\overline{CT}} = \frac{\overline{CP}}{\overline{CS}},$$

$$\text{and } \frac{y}{x+1-y} = \frac{\frac{2x-1}{3}}{\frac{x+1}{3}} = \frac{2x-1}{x+1},$$

and $y = \dfrac{2x^2+x-1}{3x}$. Combining this and the equation for y found earlier, we get

$$\frac{x^2+2x}{2x+2} = \frac{2x^2+x-1}{3x}.$$

After cross multiplying, the solution simplifies to $x^3 = 2$. Therefore, $x = \sqrt[3]{2}$. [2] □

Origami is shown to be far more powerful than compass and straightedge as a method of construction, as these problems previously proven unsolvable through the use of compass and straightedge were proved solvable through the use of origami. However, squaring the circle, or constructing a square whose area is the same as a given circle, is impossible to construct through origami because π would need to be constructed. Pi is a transcendental number, and thus not the root of a polynomial with integer coefficients (only polynomial equations with rational coefficients can be solved with origami—π is irrational). However, curved (non-flat) creases can allow for the construction of non-algebraic numbers. An approximation for π can be constructed by valley-folding a semicircle (centered at a given point A) along the long side of a strip of paper, then making an angle of 45° or less by valley creasing from one end of the semicircle (labeled B). An example of a valley fold can be seen in fig. 2.1. Once the crease is folded flat and the semicircle is folded, part of a cone should be formed. Then, the raw edge of the paper positioned by the 45° (or less) crease should slide into the semicircle. If the point where the raw edge touches the other end of the semicircle is creased and labeled C, the length of $\overline{BC}$ is the perimeter of the semicircle. Thus, if the semicircle's radius $\overline{AB} = 1, \overline{BC}/\overline{AB} = \pi$, implying $BC = \pi$ (to about two decimal places). [12]

The methods used in this construction could be formalized as follows: "Given a line L and a curve C, we can align L onto C or vice-versa." This is similar to the third Huzita axiom—"Given two lines l_1 and l_2, we can fold line l_1 onto l_2," though nothing is mentioned about creasing to place L onto C, only that they can be aligned onto each other. [12]

When discussing origami constructible numbers, a few concepts must first be defined. We will begin with two points, p_0 and p_1, and define the distance between them, $|p_0p_1|$, to be 1.

Definition 1.1. *A line l is constructible if a fold can be formed along line l.*

Definition 1.2. *A point p is constructible if two lines can be constructed that cross at point p.*

Definition 1.3. *A number α is constructible if two points can be constructed a distance α apart. [7]*

In order to establish the Cartesian plane, we must first create a vector from p_0 to p_1 as the initial basis vector e_1. Then, to find the second basis vector e_2, we construct a line segment with a magnitude equal to that of e_1 extending at a right angle from point p_0 to e_1. The endpoint of this line segment will be denoted p'_1, reaffirming the line segment's unit length 1. Thus, e_2, the second basis vector, will extend from p_0 to p'_1. [7]

Function 1. Given two points p_0 and p_1, construct a third point p'_1 a distance $|p_0p_1|$ from point p_0 such that $p_0p'_1 \perp p_0p_1$. [7]

The construction of these functions involves the use of the original seven Huzita-Justin Axioms.

Using Axiom 1, construct the line l_1 passing through p_0 and p_1.

Using Axiom 4, construct the line $l_2 \perp l_1$ passing through p_1.

Using Axiom 4, construct the line $l_3 \perp l_1$ passing through p_0.

Using Axiom 3, construct the line l_4 placing l_1 onto l_2.

The point constructed upon the crossing of l_3 and l_4 is the point p'_1. [7]

Proof. Since p'_1 lies on l_3 which, by definition, is perpendicular to l_1 and constructible by Axiom 4, $\overline{p_0p_1}$ is perpendicular to $\overline{p_0p'_1}$. The acute angle formed by l_1 and l_4 must be equal to that which is created by l_2 and l_4, since l_1 was placed onto l_2 when forming l_4. The angle created by l_1 and l_2 is bisected by l_4. Lines l_2 and l_3 are parallel, and thus both are perpendicular to l_1. By alternate interior angles, the acute angle made by l_2 and l_4 is equal to the angle formed by l_3 and l_4. Finally, since $\angle p_0p_1p'_1 \cong \angle p_0p'_1p_1, \triangle p_1p_0p'_1$ is an isosceles triangle, such that $|p_0p_1| = |p_0p'_1|$. [7] $\square$

Function 2. Given two points p_0 and p_1, a third point p_2 can be constructed such that p_2 is collinear with p_0 and p_1, and thus $|p_0p_1| = |p_1p_2|$. [7]

Use Function 1 to create p_1' from p_0 and p_1 such that $\angle p_1p_0p_1'$ is a right angle. From Function 1, l_1 (through p_0 and p_1), l_2 (through p_1 and perpendicular to l_1), and l_3 (through p_0 and perpendicular to l_1) have been formed.

Use Axiom 4 to create the line $l_5 \perp l_3$ through p_1'.

Use Axiom 3 to create the line l_6 placing l_5 onto l_2.

The point constructed upon the crossing of l_1 and l_6 is p_2. [7]

Proof. The points p_0, p_1, and p_2 are collinear, since they all lie on l_1. By congruent triangles $\triangle p_1p_0p_1' \cong \triangle p_2p_1p_3$ having three equivalent angles and one equivalent side, where the intersection of l_2 and l_5 is p_3, $|p_0p_1| = |p_1p_2|$. [7] □

All positive and negative integers have been shown to be constructible along the axes e_1 and e_2 (whose constructions are described earlier). Thus, any point in $\mathbb{Z}$x$\mathbb{Z}$ is constructible by finding the intersection of a perpendicular extended from integers constructed on each axis. We will now move from $\mathbb{Z}$x$\mathbb{Z}$ to $\mathbb{Q}$x$\mathbb{Q}$. [7]

Function 3. Given two constructible numbers α and β, we can construct $\frac{\alpha}{\beta}$, their ratio. [7]

Extending from p_0, construct a point p_a a distance α along the e_1 axis and a point p_b a distance β along the e_2 axis.

Use Axiom 4 to construct the line $l_1 \perp \overline{p_0p_a}$ through p_a.

Use Axiom 5 to create the line l_2 passing through p_0 and placing p_b onto l_1.

Use Axiom 4 to create the line $l_3 \perp l_2$ passing through p_b. Denote the intersection of l_1 and l_3 point p_2. Now $|p_0p_2| = \beta$.

Use Axiom 1 to form the line l_4 passing through p_0 and p_2.

Use Function 1 to form p_1' one unit along the e_2 axis.

Use Axiom 5 to construct the line l_5 passing through p_0 and placing p_1' onto l_4.

Use Axiom 4 to create the line $l_6 \perp l_5$ passing through p_1'. Denote the intersection of l_4 and l_6 point p_3. Thus, $|p_0p_3| = 1$.

Use Axiom 1 to form the line l_0 through p_0 and p_1.

Use Axiom 4 to construct the line $l_7 \perp l_0$ passing through p_3. Denote the intersection of l_0 and l_7 point p_r.

Thus, $|p_0p_r| = \frac{\alpha}{\beta}$. [7]

Proof. Denote the intersection of l_2 and l_3 as p_x. The lengths of $\overline{p_bp_x}$ and $\overline{p_xp_2}$ must be equivalent, as they are able to be superimposed upon each

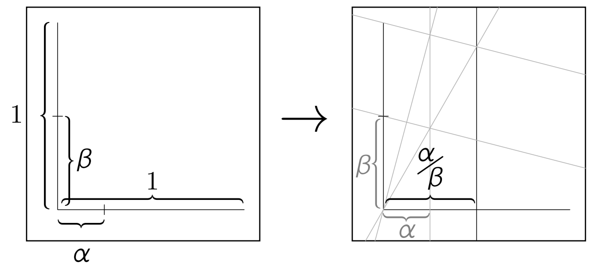


Figure 1.5: Ratio Construction and Creases Involved

other. Next, $\triangle p_0p_xp_b$ and $\triangle p_0p_xp_2$ both share the side $\overline{p_0p_x}$, thus by side-angle-side, the two triangles are congruent. Similarly, it can be proven that $|p_0p_3| = 1$. In addition, $\triangle p_0p_rp_3 \sim \triangle p_0p_ap_2$. These two triangles are right triangles, and both share an angle—thus, all three angles corresponding with one another are equal, and the triangles are similar. Therefore, the ratio of corresponding sides is also equal, and $\frac{|p_0p_a|}{|p_0p_2|} = \frac{|p_0p_r|}{|p_0p_3|}$. Since $|p_0p_a| = \alpha$, $|p_0p_2| = \beta$, and $|p_0p_3| = 1$, by substitution, $\frac{\alpha}{\beta} = \frac{|p_0p_r|}{1} = |p_0p_r|$. [7] $\square$

Thus, by extending a perpendicular from the intersection of a ratio of integers constructed on each axis, any point in $\mathbb{Q}^2$ can be constructed [7]. It has been proven that the division of any two constructible numbers is possible. Since it is apparent we are able to construct the rational numbers, we will continue to prove that origami constructible numbers form a field. The rationals form a field, and thus all we know how to construct forms a field. To ensure the structure of the field is maintained even if constructible non-rational numbers are discovered, we must determine if the origami constructible numbers are closed under addition, subtraction, and multiplication, along with the previously proven division. [7]

Function 4. Given two constructible numbers α and β, we can construct their sum $\alpha + \beta$ or their difference $\alpha – \beta$. [7]

Create the point p_a extending from p_0 a distance α along the e_1 axis.

Create the point p_b extending from p_a a distance β along the e_1 axis in the same direction as e_1 for addition and in the opposite direction for subtraction.

Ultimately, $|p_0p_b| = \alpha + \beta$ (or $\alpha – \beta$). [7]

Proof. The appropriate sum or difference has obviously been constructed if any previously constructible points based from p_a can indeed be constructed.

To construct the necessary unit length based from point p_a, begin by constructing the axes using Axiom 1 to form the line $l_1 = \overline{p_0 p_a}$ and the line $l_2 = \overline{p_0 p_1'}$. Therefore, l_1 is perpendicular to l_2.

Use Axiom 4 to construct the line l_3 perpendicular to l_2 through point p_1', and again to form the line l_4 perpendicular to l_1 through p_a. Denote the intersection of l_3 and l_4 point p_a'.

Use Axiom 3 to construct the line l_5 placing l_3 onto l_4. Denote the intersection of l_1 and l_5 point p_2.

The lines l_1 and l_3 are one unit apart and the triangle created is isosceles (by the proof of Function 2)—thus, p_2 is one unit away from p_a. The number β previously constructible from p_0 using p_1 is now constructible using p_a and p_2. [7] □

Function 5. Given two constructible numbers α and β, we can construct $\alpha\beta$, their product. [7]

Because of the repetitive nature of these constructions, these steps will not be listed here. All that must be done to construct the multiplication of α and β is divide α by the reciprocal of β—the construction of such division has already been proven. Using Function 3, we divide one by β to get $\frac{1}{\beta}$, then divide α by $\frac{1}{\beta}$. After simplification, the result is equal to $\alpha\beta$. [7]

It has been proven that the set of constructible numbers is closed under addition, subtraction, multiplication, and division—thus, it can be concluded that the set of constructible numbers forms a field. Ultimately, the field of origami constructible numbers is closed under taking both square roots and cube roots. The construction of the square root of any constructible number implies the field of origami constructible numbers contains the field of compass and straightedge constructible numbers. For convenience, the steps and proofs of these constructions will not be included, but referring to Shepherd Engle's paper entitled "Origami, Algebra, and the Cubic" will give a thorough understanding of the constructions and their corresponding proofs. The originally unsolvable (through compass and straightedge) problems of trisecting an angle and doubling the cube, as described earlier, were discovered to be easily solvable through origami. This is due in part to trisecting an angle reducing to solving a cubic equation, and doubling the cube reducing to constructing the cube root of two [7].

A good route to pursue in the future would look at the efficiency of origami constructions versus other construction tools. Problems relating to this topic include the study of the existence of constructions being algebraically and geometrically optimal. Determining the uniqueness and optimality of a given construction remains an open problem. In addition, one may consider forming relations satisfied by the degrees, levels, and orders of constructions [21].

1.2 Lill's method

In 1868, Eduard Lill published a geometric method for finding real roots of polynomials. Margherita Beloch realized, in 1936, that this method could be adapted to paper folding, allowing one to find roots of polynomials with rational coefficients by making a series of folds [13]. With single-fold axioms, polynomials of degree 3 or less can be solved. In general, if we allow n simultaneous folds, we can solve polynomials of degree up to $n+2$ [13]. See [22] for more information on Lill's Method adapted for origami.

Before looking at Lill's method, we must learn about *Horner's form* of a polynomial. For a polynomial written in standard form as

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots a_1 x + a_0,$$

Horner's form is

$$p(x) = (((a_n x + a_{n-1})x + a_{n-2})x + \cdots)x + a_0.$$

Horner's form leads to an efficient algorithm for computing values of a polynomial. For finding the value $p(x_0)$, compute a series of values:

$$\begin{aligned} b_n &:= a_n \\ b_{n-1} &:= b_n x_0 + a_{n-1} \\ &\vdots \\ b_0 &:= b_1 x_0 + a_0. \end{aligned}$$

Then $b_0 = p(x_0)$.

This algorithm provides the basis for Lill's method. We'll start with an example of a cubic equation; consider

$$f(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_0.$$

We will represent this graphically with a right-angled path, shown in figure 1.6(a). To construct the path, start at the origin and walk a distance a_3 vertically. Then turn clockwise 90° and walk a distance a_2. (Note that if a_2 is negative, you will walk backwards.) For each new coefficient, repeat this process of turning 90° and walking for a distance given by the coefficient.

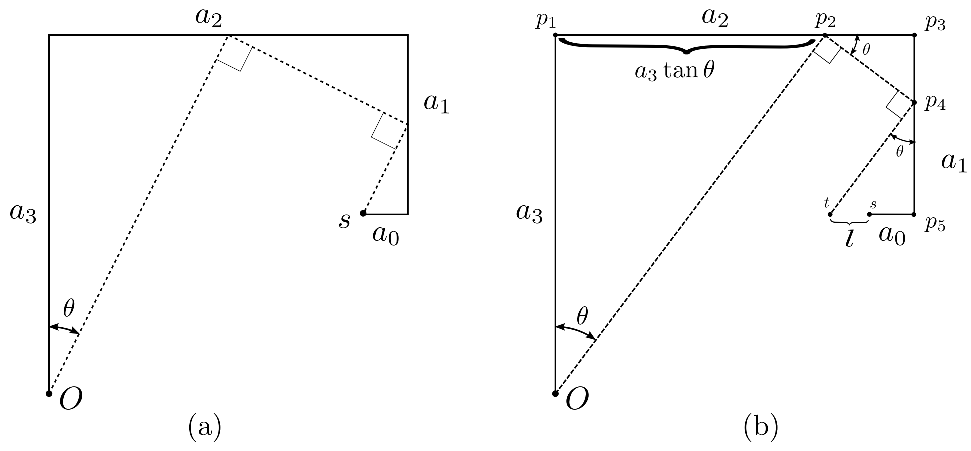


Figure 1.6: The outer right-angled path has lengths determined by the coefficients. In (a), the endpoints of the inner and outer paths coincide. In (b), they do not.

Now, create a second right-angled path, represented in 1.6(a) as a dotted line starting at the origin O. Launch the line at an angle θ and any time you intersect the original right-angled path, reflect at an angle of 90°. Now, vary the angle θ until the endpoints of both paths coincide at s. When the endpoints coincide, $-\tan\theta$ is a root of f; so $f(-\tan\theta) = 0$.

Why does this occur? Let's consider the length l between points s and t in fig. 1.6(b). In order to find this length, first note that the triangles Op_1p_2, $p_2p_3p_4$, and p_4p_5t are all similar. Now we'll find the length of segment $\overline{p_1p_2}$. Simple trigonometry tells us that it is $a_3 \tan\theta$. Next, the length of $\overline{p_2p_3}$ is $a_2 - a_3 \tan\theta$.

Now, let's find the length of $\overline{p_4p_5}$. In a similar manner to above, we learn that it is

$$a_1 - (a_2 - a_3 \tan\theta) \tan\theta.$$

And similarly, the length of l is

$$a_0 - (a_1 - (a_2 - a_3 \tan\theta) \tan\theta) \tan\theta.$$

Note that when this length is 0, we have

$$0 = a_0 - (a_1 - (a_2 - a_3 \tan\theta) \tan\theta) \tan\theta$$

and we have constructed Horner's form of the polynomial f, and so when s and t coincide, $-\tan\theta$ is a root of f!

How can we adapt this method to folding? It is fairly simple. We define a unit length on the paper to be the length between some points p_1 and p_2.

As long as the coefficients are rational (or in general algebraic if we allow simultaneous folds), we can construct segments with the lengths determined by the coefficients. Thus we can construct the outer right-angled path.

Note that for a polynomial

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots a_1 x + a_0,$$

we can divide it by a_n without changing the roots, giving us the monic polynomial

$$p(x) = x^n + \frac{a_{n-1}}{a_n}x^{n-1} + \cdots \frac{a_1}{a_n}x + \frac{a_0}{a_n}.$$

Using a monic polynomial with origami has the advantage that the first segment in the outer right-angled path has unit length. This means, in particular, that the length $\overline{p_1p_2}$ will (up to sign) be a root of the polynomial, since

$$\overline{p_1p_2} = a_n \tan\theta = \tan\theta.$$

So solving a polynomial with origami means constructing segments with length equal to the (real) roots of the polynomial. Constructing the outer right-angled path is simple using Axiom 4, which allows us to construct lines perpendicular to lines we have already. Creating the inner path is more involved. In the case of a cubic equation, using Axiom 6 will be necesary to find the proper lines to create the path. Once we have created the inner right-angled path, we have a length equal to a root of the polynomial.

2 General Foldability

The above constructions, for the most part, rely on making a single crease and then unfolding it, to use the mark it left on the paper. However, most origami does not work this way - after making a fold, it stays, and then we fold over it. Here we discuss the mathematical formalism for this kind of folding.

One of the best ways to study origami is through studying crease patterns. Intuitively, a crease pattern is obtained by folding an origami model, and then unfolding the paper. The lines left on the paper by the folds form the crease pattern. We will only consider crease patterns that use straight-line creases. While it is possible to fold origami using curved creases, these types of models are much less common and are more difficult to study mathematically and curved creases will not appear in any of the topics we consider.

Definition 2.1 (Crease Pattern)**.** *A* crease pattern *is a pair (C, R) where C is a set of line segments that intersect only at their endpoints and R, representing the piece of paper, is a closed connected subset of $\mathbb{R}^2$, s.t. $C \cap R = C$. An intersection of creases is called a* vertex*. Note that intersections occurring on the boundary of R will not be considered vertices.*

This definition of a crease pattern is inspired by [23]. For an example of a crease pattern, see fig. 2.2(a), which is the crease pattern for the common bird base form.

Generally, we will consider origami using a square piece of paper paper.

Note that a crease pattern alone does not capture all of the relevant information about an origami model. Every crease can fold upwards or downwards. These are called valley folds and a mountain folds. We depict valley folds with dashed lines and mountain folds with dash-dotted lines, as shown in fig. 2.1.

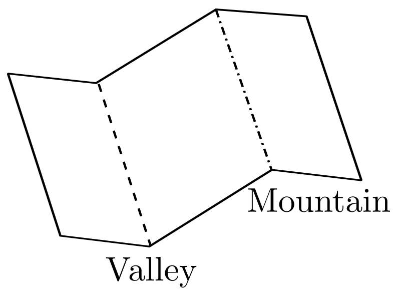


Figure 2.1: A mountain fold with a dash-dotted line, and a valley fold with a dashed line.

Definition 2.2. *A* ***mountain-valley assignment*** *of a crease pattern $C = \{V, E\}$ is a many-to-one mapping $E \to \{M, V\}$.*

Fig. 2.2(a) is the crease pattern for the bird base, and (b) is the bird base with a mountain valley assignment.

Applying a different mountain-valley assignment to the same crease pattern leads to a different folded form. Note that for a given crease pattern, some assignments may by (flat) foldable while others may not be.

When we fold an origami model, we don't rip the paper or stretch it. In order to capture this notion, we define a folding.

Definition 2.3 (Folding)**.** *A* folding *of a crease pattern (C, R) is an assignment of angles to each crease, such that the paper, when folded with the*

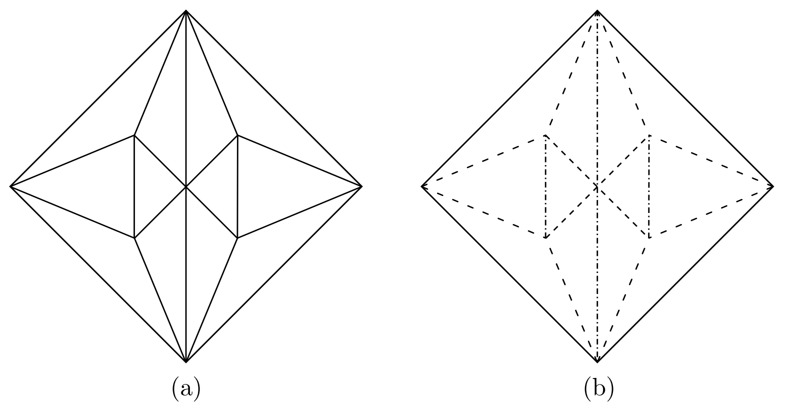


Figure 2.2: A bird base crease pattern, both assigned and unassigned.

angles at the creases, does not rip or stretch, and the faces of the final form are flat.

In general, we consider folding to take place in an arbitrary number of dimensions in order to avoid the paper self-intersecting. Sometimes, as in fig 2.3, we can have a folding in three dimensions where the paper self-intersects. This does not correspond to how paper behaves in real life. To avoid avoid this problem we define a valid folding.

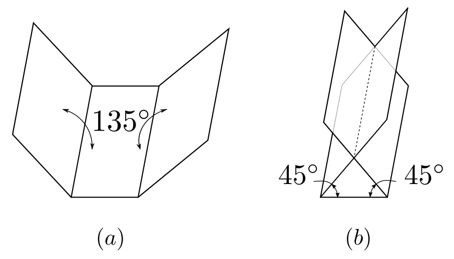


Figure 2.3: A valid folding on the left, and invalid on the right, for the same crease pattern.

Definition 2.4 (Valid Folding)**.** *A* valid folding *is a folding that does not cause the paper to self-intersect, when projected into* $\mathbb{R}^3$ *by ignoring all higher dimensions.*

See fig. 3.5 for an example of a crease pattern that has no valid folding, but that can be folded flat if the paper is allowed to self-intersect.

Note that the definitions of folding and valid folding do not apply to flat foldings, but only to foldings in $\mathbb{R}^n$.

2.1 Foldings and Knot Theory

The following theorem was inspired by a comment made by Jason Ku (MIT) in his presentation of an origami hole-filling algorithm at JMM 2016. Asked if his algorithm dealt with self-intersection, he said it did not, but that there are a couple easy initial checks one can perform. "For instance," he pointed out, "the border of the paper must map to the unknot."

This theorem ties together knot theory and origami, and will hopefully allow us to gain further insight into ways in which foldings of crease patterns may or may not self-intersect. An introduction to knot theory sufficient for our purposes can be found in [18].

Theorem 2.5. *A crease pattern with a given folding is valid if and only if every Jordan curve drawn on the unfolded paper gets mapped to the unknot in the final folded form.*

Proof. First we need to show that if a folding is valid, then no Jordan curve becomes a knot. A folding is valid when the paper has no self-intersection in $\mathbb{R}^3$. Any time we make a fold, changing the positions of the faces of the paper may introduce crossings into the Jordan curve. However, as long as the fold does not cause the paper to self-intersect, these crossings can be undone with Reidermeister moves.

The more interesting direction is showing that if a folding for a crease pattern is invalid, then there exists a Jordan curve on the unfolded paper that gets mapped to a non-trivial knot on the final folded form.

If a folding is invalid, that means that the paper intersects itself. This intersection must occur along a line segment, because the faces of the paper are flat. Thus, every intersection locally looks like fig. 2.4. Because the paper is connected, we can draw a path from one end of one plane to the other, as shown in fig.2.4(a). We can thicken the path into a ribbon of paper by an ϵ amount, shown in fig. 2.4(b). If the ribbon intersects itself, there are two possible cases. The first is that the path intersects itself before being thickened into a ribbon, as shown in 2.5(a). In this case, we can nudge the path a little bit so that the path no longer intersects itself. Now we may find ourselves in the second case, in which the ribbon partially intersects itself, as in 2.5(b). Take the line segment along the intersection, and let δ be the length along which the intersection actually occurs. Now we can simply pick a small enough $\epsilon \leq \frac{1}{2}\delta$ so that there is no intersection anymore. The ribbon without intersection at all is in 2.5(c).

Now we have a ribbon connecting the intersecting planes, and we draw the curve illustrated by 2.6(a) along the ribbon. If the paper does not twist

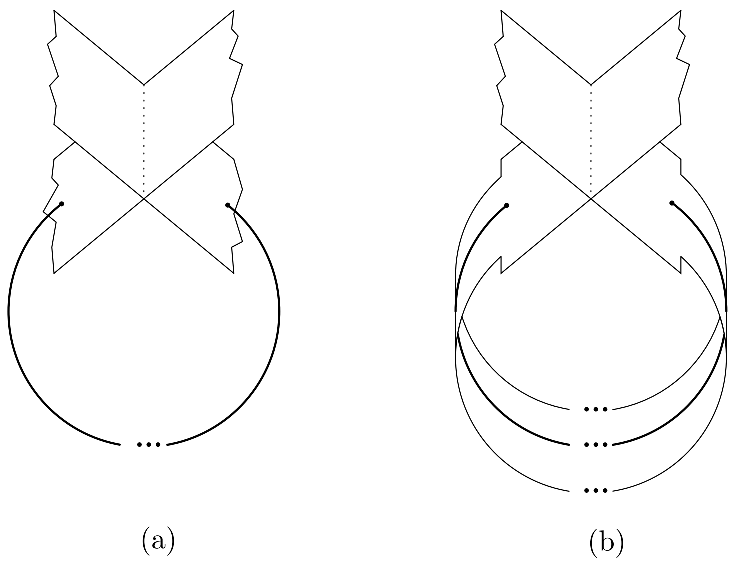


Figure 2.4: (a) Finding a path between the intersecting planes. (b) Widening the path into a ribbon.

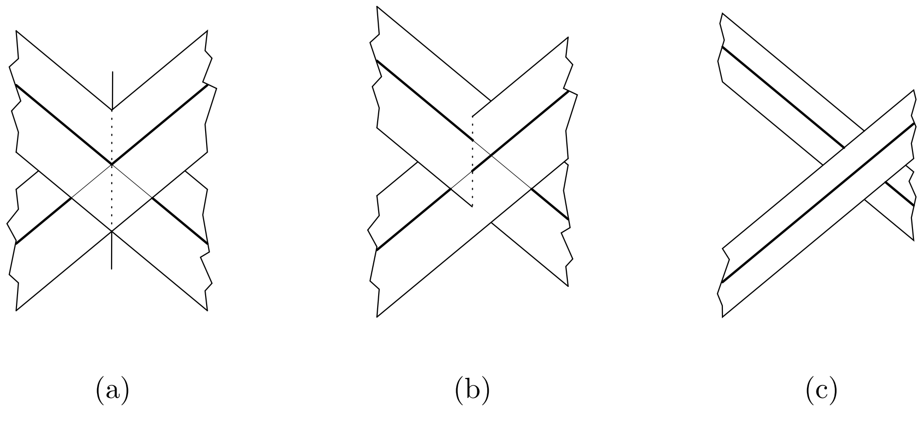


Figure 2.5: (a) The path self intersects, so the ribbons intersect. (b) The path does not intersect, but the ribbon does. (c) The path no longer intersects

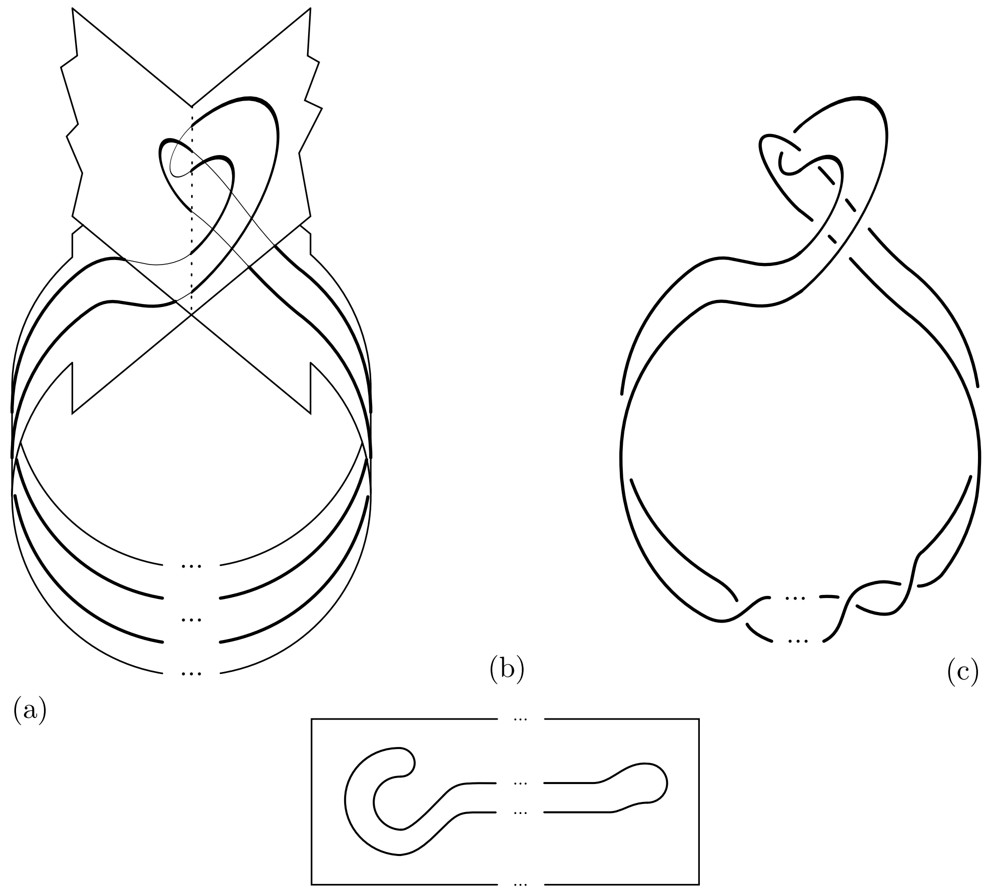


Figure 2.6: (a) The self-intersecting paper with our special jordan curve embedded on it. (b) The curve shown on a flat paper in the unfolded form. (c) The curve projected as a knot diagram.

at all, this produces a trefoil knot. However, depending on how the paper twists between the two ends of the ribbon, the curve might not actually produce a nontrivial knot. Look at fig. 2.6(c) for an example of the curve with twists. Note that consecutive twists in opposite directions cancel each other out. After using Reidermeister moves to omit consecutive twists in opposite directions from our diagram, we will end up with a certain number of over- or under-crossings. We may need to reverse the direction of the crossings in the determined part of our construction in order for the final diagram to describe a nontrivial knot. If our construction gets mapped to the unknot simply reflect fig. 2.6(b) and it will produce a knot. □

Because there are many ways to characterize knots (e.g. with polynomials, groups, &c.), this theorem might allow us to adapt some of these to create a characteristic for crease patterns or foldings. We might be able to use this to describe the complexity of a crease pattern by finding, for example, the number of ways it has to self-intersect for a given folding.

3 Flat Foldability

We've been looking at questions of general foldability - is there a way to fold a crease pattern at all? Here, we focus on one of the biggest questions of origami: when can a crease pattern be folded flat? Intuitively, a flat model is one that can be pressed between the pages of a book without adding any new creases.

We know some necessary conditions for flat foldability. Unfortunately, these conditions are only sufficient in the case of a single vertex.

3.1 Single Vertex Conditions

The two most important tools for determining flat foldability are Kawasaki's and Maekawa's theorems. These deal with the angles between creases and the mountain-valley assignment of a crease pattern, respectively. We'll start with Kawasaki's theorem, which in the case of a single vertex is a sufficient condition (as long as we have a valid MV-assignment). Note that the number of creases around a vertex must be even for Kawasaki's theorem to apply. Information on these theorems can be found in [14] and [11].

Theorem 3.1 (Kawasaki's Theorem)**.** *The alternate angles formed by creases around a vertex must sum to* $180°$ *in order for the vertex to fold flat.*

Proof. Consider an ant walking around the perimeter of the crease pattern in Fig. 3.1. The ant first walks a distance a, then turns around and walks a distance b, continuing in this manner until it comes back to where it starts at O. So we have

$$a - b + c - d + e - f = 0$$

or

$$a + c + e = b + d + f.$$

The theorem is proved in an identical manner for a crease pattern with more edges adjacent to the single central vertex. □

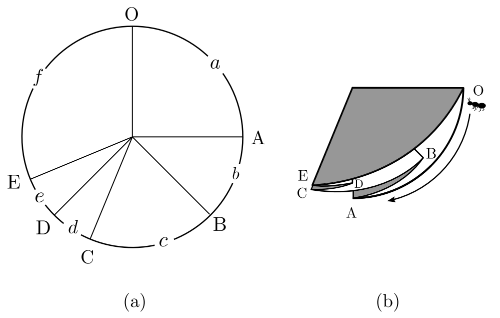


Figure 3.1: An ant crawling around the perimeter of a crease pattern.

See fig. 3.2 which shows a crease pattern that is foldable by Kawasaki's theorem, and next to it one that is unfoldable.

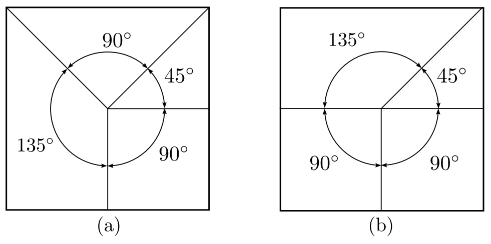


Figure 3.2: These crease patterns are valid and invalid, respectively.

Now, we have a purely geometric condition. However, Kawasaki's theorem does not include any information about assigned crease patterns. To take into account this information, we have Maekawa's theorem.

Theorem 3.2 (Maekawa's Theorem). *Let M be the number of mountain folds and V be the number of valley folds; if the vertex folds flat, then $M - V = \pm 2$.*

Proof. Consult fig. 3.3. The total number of creases is $M+V$. Consider the flat-folded vertex. Without loss of generality, let each valley crease represent a turn of $180°$ and each mountain crease represent a turn of $-180°$. Then as you walk around the vertex, when you get back to the point where you started you will have crossed every crease, and will have made a total $360°$ turn. Then

$$\begin{aligned} 180V - 180M &= 360 \\ V - M &= 2 \end{aligned}$$

□

See fig. 3.4, which shows a crease pattern with 4 valley folds and 2 mountain folds (valid) and 3 each (invalid).

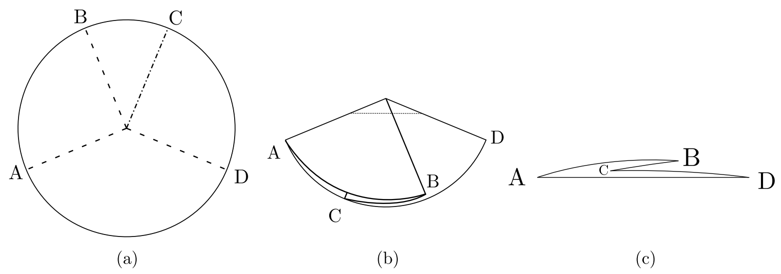


Figure 3.3: Folding up (a) produces (b). Cutting (b) at the line produces the cross-section (c).

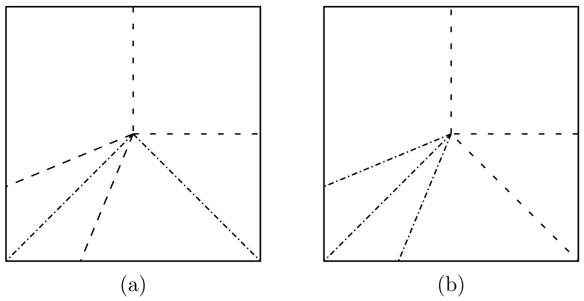


Figure 3.4: A crease pattern with a valid and invalid mountain-valley assignment.

Definition 3.3 (Local Flat Foldability)**.** *A vertex is* locally flat foldable *when it satisfies Kawasaki's Theorem and it has a mountain-valley assignment that satisfies Maekawa's theorem.*

3.2 Multiple Vertex Crease Patterns

Note that a crease pattern can be locally flat foldable at every vertex but still be globally unfoldable. A crease pattern is globally flat foldable if it can actually be folded flat without ripping or causing the paper to self-intersect, or introducing new creases not in the crease pattern. The crease pattern in fig. 3.5 has no valid mountain-valley assignment, and so cannot be globally flat folded, although every vertex in the pattern satisfies Kawasaki's theorem and can be given an assignment that satisfies Maekawa's theorem.

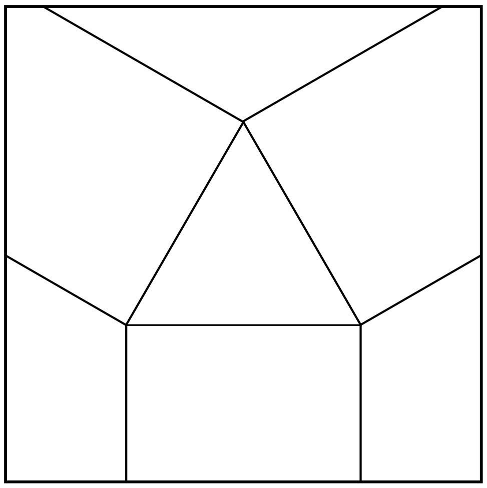


Figure 3.5: An unfoldable crease pattern.

Definition 3.4. *A crease pattern is locally flat foldable when every vertex in the crease pattern is locally flat foldable.*

We can sometimes determine if a locally flat foldable crease pattern is not globally flat foldable by giving it a mountain-valley assignment and seeing if the assignment has inconsistencies. This is how we should that the pattern in fig. 3.5 is not flat foldable. Without loss of generally, w assign one of the central creases to be a mountain fold. Then, we can determine that the crease clockwise around the triangle must be valley; then the next must be mountain; and this forces our original crease to be valley. But this is an inconsistency, and thus the crease pattern cannot be flat folded.

In general, the task of determining flat foldability for patterns with multiple vertices is very difficult. There are ways to simplify the problem: for example, we can consider classes of simple crease patterns, such as those with only evenly spaced horizontal and vertical creases. This type of pattern, known as a map, will be discussed below.

4 Computational Folding Questions: An Overview

4.1 Basics of Algorithmic Analysis

We will now turn our attention to computational questions relating to origami. One could ask, for example, whether or not it is possible to write an algorithm to determine whether a specific assigned crease pattern is flat-foldable. If such an algorithm exists, what is the fastest such algorithm? Before we are able to answer these questions, we must first briefly cover relevant concepts from theoretical computer science.

The most important concept in theoretical computer science is that of an algorithm. An *algorithm* is a set of instructions that can be followed to accomplish a task. Suppose, for example, that we are given a finite set S of integers and a target integer k, and told to determine whether or not there exists a subset $R \subseteq S$ such that the sum of all the members of R is equal to k. (This is known as the SUBSET-SUM problem.) One possible algorithm to solve this problem is as follows: "For every possible subset R of S, compute the sum of the members of R. If this sum is equal to k, return `true`. After we've checked every subset, return `false`."

Observe first that this algorithm will always return the correct answer. If some subset R of S has the sum of its members equal to k, then we will find it since we check every subset of S. Therefore if such a subset exists, we return true. If no subset of S has the sum of its members equal to k, then we will never return `true`, and thus after we've checked every subset, we will return `false.` Therefore if such a subset does not exist, we will return `false`. This means we return `true` if and only if some subset of S has the sum of its members equal to k. This means our algorithm indeed tests exactly what we hoped it would, and always returns the correct answer!

Since we know that our algorithm will always find the right answer, we might wonder how long it will take to do so. This question requires us to develop some formal way of discussing the time an algorithm takes to run. In computer science, we measure the running time of an algorithm in terms of the number of "operations" it must perform *in the worst case.* The precise definition of an operation is beyond the scope of this paper, but for our purposes we will consider most basic arithmetic to be single operations (addition, subtraction, multiplication, checking equality, etc.)

Note that we are interested in the worst case running time of an algorithm because it provides an upper bound for the running time. Every time we run the algorithm, it must perform *at most* as many operations as in the worst case. Therefore worst-case analysis of an algorithm allows us to have a guarantee of the speed of our algorithm.

We can compute the exact number of operations our algorithm will perform in the worst case. Note that we look at 2^n total subsets of S in the worst case, where we must look at every one, and each subset has at most n elements. It takes $n-1$ additions to find the sum of n numbers, and one comparison to test if this sum is equal to k. Therefore we are doing at most n operations for each of the 2^n subset of S, and thus we are doing at most $n * 2^n$ operations in total. Therefore the worst-case running time of our algorithm is $n * 2^n$ operations.

Computer scientists use "big-O notation" to discuss the asymptotic running time of algorithms. We wish to know what happens to the running time of an algorithm as the size of the input gets very large. Big-O notation basically encodes the answer to the question "what is the size of the most significant term of our running time?" Rather than give the formal definition of big-O notation, we give a table of example functions $f(n)$ and their corresponding big-O notation representation $\mathcal{O}(f(n))$, where k is a constant.

$f(n)$	$\mathcal{O}(f(n))$	Polynomial time?
n	$\mathcal{O}(n)$	Yes
$3n$	$\mathcal{O}(n)$	Yes
$n+2$	$\mathcal{O}(n)$	Yes
$5n-3$	$\mathcal{O}(n)$	Yes
$3n+4n^2$	$\mathcal{O}(n^2)$	Yes
$2n^4-3n^2+\sqrt{6n}$	$\mathcal{O}(n^4)$	Yes
$a_k n^k + \cdots + a_1 n + a_0$	$\mathcal{O}(n^k)$	Yes
$2^n + n^k$	$\mathcal{O}(2^n)$	No
$n * 2^n + 2^n$	$\mathcal{O}(n * 2^n)$	No
n^n	$\mathcal{O}(n^n)$	No (!)
$n! + 2^n$	$\mathcal{O}(n!)$	No
k	$\mathcal{O}(1)$	Yes

An algorithm is said to run in time $\mathcal{O}(f(n))$ if its worst-case performance on an input of size n takes $\mathcal{O}(f(n))$ operations. Recall that in the worst case our algorithm for the Subset-Sum problem performed $n * 2^n$ operations. This means our algorithm runs in time $\mathcal{O}(n * 2^n)$. There may be particular instances of this problem which our algorithm finishes in $\mathcal{O}(2^n)$ operations or even in $\mathcal{O}(n)$ operations. However, we now have a guarantee that our algorithm will always finish in fewer than $\mathcal{O}(n * 2^n)$ operations.

An algorithm is said to run in *constant time* it its worst-case running time is $\mathcal{O}(1)$. An algorithm is said to run in *linear time* it its worst-case running time is $\mathcal{O}(n)$. An algorithm is said to run in *polynomial time* if its worst-case running time is $\mathcal{O}(n^k)$ for some constant k. An algorithm is considered efficient (for our purposes) if it runs in polynomial time (or better).

Note that our algorithm for the Subset-Sum problem was *not* a polynomial time algorithm. One might wonder whether or not there exists a polynomial time algorithm which solves the Subset-Sum problem. This question lies at the core of computational complexity theory, and will be addressed in the next section.

4.2 Introduction to Computational Complexity Theory

We will now develop a cumbersome framework for discussing the difficulty of problems, in order to make meaningful statements about the difficulty of various origami-related problems. We will borrow techniques from computational complexity theory, which attempts to classify problems into different classes based on their relative difficulty.

One important complexity class is P, the class of all problems which can be solved in polynomial time. For example, consider the Subset problem, where we are given the task of determining whether a finite set S of size n contains a finite set R as a subset. This problem can be solved in polynomial time by the following algorithm: "For each member r of R, check if $r \in S$ by checking if $r = s$ for each $s \in S$. If some $r \in R$ is not in S, return `false`. After we check every member of R, return `true`."

We can also ask whether the solution to a particular problem can be verified as correct in polynomial time. Given an instance p of a particular problem, and a proposed solution s, we can ask what the complexity of determining the correctness of our solution is. For example, a proposed solution to the Subset-Sum problem would consist of the subset that sums to k. Given this subset, we can determine whether or not it is indeed a subset in polynomial time (as above), and whether its members indeed sum to k in $|R| < |S| = n$ operations. Therefore we can determine the correctness of a solution to Subset-Sum in polynomial time. The class of all problems whose solutions can be verified in polynomial time is called NP. Perhaps surprisingly, not every problem is in NP (examples are beyond the scope of this paper).

Clearly every problem in P is also in NP. Suppose we have a problem in P and a proposed solution to the problem. We can just solve the problem in polynomial time (since it's in P), and compare the solution to the one we were trying to verify. If we got the same answer, then our given solution was correct. If we get a different answer, then our given solution was wrong. Note that this still works if there were multiple possible valid solutions to the problem, but requires a definition detail which is beyond the scope of this paper.

The most important open problem in computer science is the problem of determining whether or not P=NP. This problem is one of the Millennium Prize Problems (along with the Poincare Conjecture and the Riemann Hypothesis), meaning there is a \$1 million prize for the person who solves it. There has been a lot of study surrounding this problem which has led to the development of various subfields of computer science.

How would one go about determining whether or not P=NP? Since we already know P $\subseteq$ NP, determining whether P=NP reduces to determining whether NP $\subseteq$ P. This is equivalent to determining whether every problem in NP can be solved in polynomial time. Intuitively, this would mean that solving a problem is as difficult as verifying a solution to the problem.

Now, how would one go about determining whether NP $\subseteq$ P? If we could find a problem harder than any problem in NP that can be solved by a polynomial time algorithm, then we would know that every problem in NP can be solved by a polynomial time algorithm. A problem that is at least as hard as every problem in NP is creatively called *NP-hard.* On the other hand, if we want to show that P $\subset$ NP, we would need to find a problem in NP and prove that there is no polynomial time algorithm that solves it. An NP-hard problem that is in NP is called *NP-complete.* An NP-complete problem has a polynomial time solution if and only if P=NP.

At the end of the previous section, we posed the question of whether or not there exists a polynomial time algorithm for solving the SUBSET-SUM problem. A well-known result of complexity theory is that SUBSET-SUM is NP-complete. This means that determining whether or not there exists a polynomial time algorithm for SUBSET-SUM is exactly the \$1 million problem of determining whether P=NP.

4.3 Computational Complexity of Flat Foldability

Now that we have laboriously laid the foundations of complexity theory, we can return to the world of origami. We will state two known complexity results given by [5] without proof, since detialing a proof of NP-hardness is far beyond the scope of this paper.

General Flat Foldability is the problem of determining, for a particular unassigned crease pattern, whether or not there exists an assignment for which the crease pattern is flat foldable. General Flat Foldability is NP-hard.

Assigned Flat Foldability is the problem of determining, for a particular assigned crease pattern, whether or not the crease pattern is flat foldable. Assigned Flat Foldability is NP-hard.

In the next section, we will look at a simplified version of the above problems whose complexity is currently unknown.

5 Map Folding: A Computational Problem

5.1 Introduction to Maps

In the previous section, we proved that the question of determining assigned flat foldability is difficult in general. In the general case, our crease pattern could take any form. However, what if we restrict the problem to determining which crease patterns of a specific form are flat foldable?

For example, consider the Stamp Flat Foldability problem. Suppose we know that our crease pattern consists of n unit square faces in a single row. (We call this crease pattern a *stamp* of length n.) We would like to find an algorithm which determines whether or not a particular mountain-valley assignment of a stamp of length n is flat-foldable. It turns out that *every* assignment is flat foldable, and therefore the constant time algorithm "Yes, it is flat foldable" suffices for this case.

Claim 5.1. *Every assignment of a stamp of length n is flat foldable.*

Proof. We proceed by induction on n, the length of the stamp. For $n = 1$, we start with a flat folded form. For $n = 2$ there is only one crease, and therefore there are two possible assignments. If the crease is a mountain, then we can fold the right face under the left face. If the crease is a valley, then we can fold the right face over the left face. In either case, we have flat folded the stamp. Consider the general case of a stamp of length n. As in the $n = 2$ case, the right-most face f will either fold above or below (depending on the assignment of the crease) the face to the left of it, e. Once it has been folded, whenever we move e, we can move f since they lie on top of one another with nothing between them. We can treat our new map as if f doesn't exist. Therefore by folding f, we have created a stamp of length $n - 1$. By the inductive hypothesis, we can flat fold any stamp of length $n-1$, and thus we can flat fold this stamp. Therefore every stamp of length n is flat foldable. $\square$

This problem is certainly easier than the general case of the assigned flat foldability problem, since it has a constant time algorithm. In this section, we will explore an open question about the complexity of a slightly more complicated crease pattern called a map.

An $m \times n$ *map* is a crease pattern, along with a mountain-valley assignment, whose faces are all squares of unit side length which form an m by n grid (Note that a stamp is just a $1 \times n$ map.) Map Flat Foldability is the problem of determining whether a given $m \times n$ map is flat foldable. The computational complexity of Map Flat Foldability is currently an open problem.

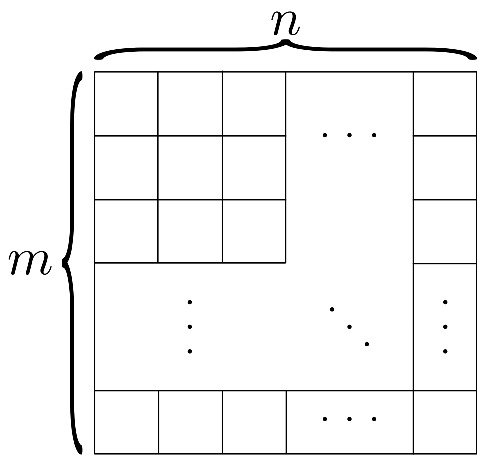


Figure 5.1: An example of a $m \times n$ map.

Not only do we know exactly where our creases are, but we also know what our flat-folded form will look like for *any* assignment. Consider the flat-folded form of a stamp of size n. We will prove that the projection of this flat-folded form onto the plane is always the same, and is exactly the unit square.

Claim 5.2. *The projection of every flat-folded stamp of length n onto the plane is the unit square.*

We can make an identical claim about the flat-folded form of a map. In proving the next claim, we will also prove the claim for stamps since a stamp is just a $1 \times n$ map.

Claim 5.3. *The projection of every flat-folded $m \times n$ map onto the plane is the unit square.*

Proof. Suppose for the sake of contradiction that the projection of the flat-folded form onto the plane is not the unit square. Then there must be some face which does not lie completely inside the unit square. No face can lie partially inside the unit square, since all of our creases are orthogonal and all of our faces are unit squares. This means some face lies completely outside of the unit square. Therefore somewhere in our flat-folded form we have two adjacent faces on the same plane. But these faces have a crease between them that hasn't been folded, and therefore this is not a flat-folded form as we supposed. ☐

According to the above claim, the flat-folded form of a map is always just a linear stack of the faces. This means that the ordering of these faces uniquely determines the flat-folded form. A linear ordering of the faces of an $m \times n$ map $\mathcal{M}$ is *valid* if it is the linear ordering of some flat folding of $\mathcal{M}$. Note that each map may have many different linear orderings corresponding to different ways to flat-fold the map. For example, all of the possible linear orderings of the faces of a stamp are valid. If there are no valid linear orderings of the faces of map, the map is not flat foldable. The question of determining whether or not a map $\mathcal{M}$ is flat foldable is equivalent to asking whether or not there exists a valid linear ordering of the faces of $\mathcal{M}$. We must therefore develop a method for testing whether or not a given linear ordering is valid.

5.2 Testing the Validity of a Linear Ordering

In this section, we will follow the work of Nishat and Whitesides in [19] to construct a linear time algorithm for determining whether or not a given linear ordering of faces is valid for a particular map.

Suppose the paper is colored dark on one side and light on the other. We can ask for a particular $m \times n$ map $\mathcal{M}$, whether a particular face is light-side or dark-side up in a particular linear ordering L. (Suppose the top-left-most face is light-side up.) The *checkerboard pattern* of a map $\mathcal{M}$ is the unique assignment of the faces of $\mathcal{M}$ to the set {light, dark} such that the top-left-most face of $\mathcal{M}$ is light and adjacent squares have differing assignments.

For every pair of adjacent faces in a map, we can use our knowledge of the checkerboard pattern and the map's assignment to determine which face in the pair is earlier in any linear ordering of the faces. For adjacent faces u

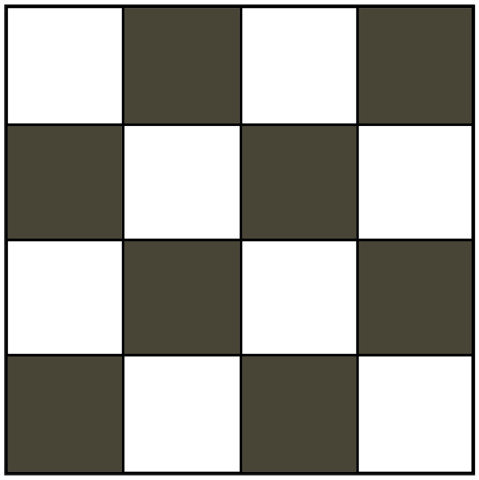


Figure 5.2: An example of a checkerboard pattern for a 4 map.

and v in a map $\mathcal{M}$, we say u *precedes* v if (i) u is light, v is dark, and the crease between u and v is a mountain or (ii) u is dark, v is light, and the crease between u and v is a valley.

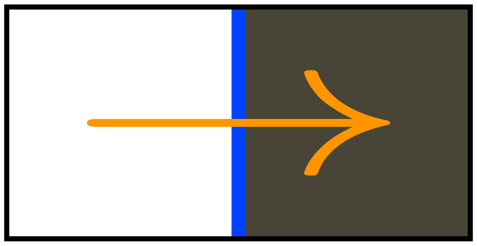


Figure 5.3: The precede relation we learn from a mountain fold.

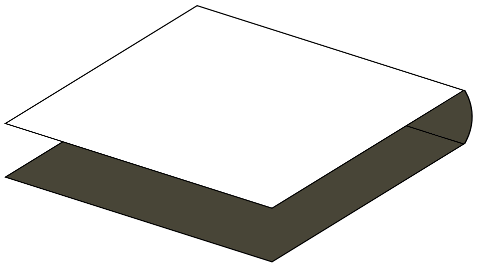


Figure 5.4: Folded mountain crease.

Lemma 5.4. *If u precedes v, then u comes before v in the linear ordering of every valid flat folding of $\mathcal{M}$.*

Proof. Suppose the crease between u and v is a mountain. Then as seen in fig. 5.3, the light-side up face comes before the dark-side up face, meaning u comes before v if the u is light-side up, v is dark-side up, and the crease between them is a mountain. This proves case (i).

Suppose the crease between u and v is a valley. Then as seen in fig. 5.5, the dark-side up face comes before the light-side up face, meaning u comes

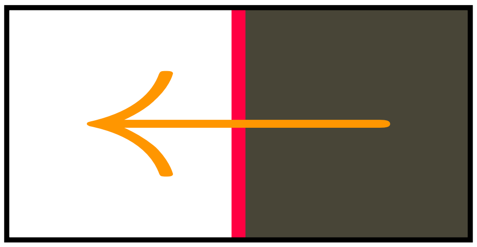


Figure 5.5: The precede relation we learn from a valley fold.

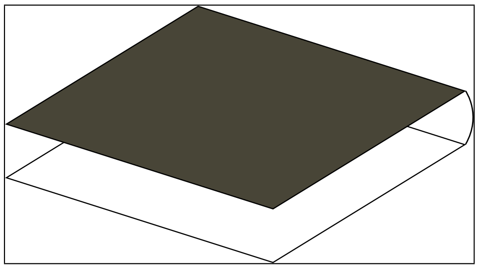


Figure 5.6: Folded valley crease.

before v if the u is dark-side up, v is light-side up, and the crease between them is a valley. This proves case (ii). $\square$

We can induce a partial ordering $\mathcal{P}$ on the faces of $\mathcal{M}$ by saying that $u < v$ if u precedes v. This partial ordering must be satisfied in any valid linear ordering of faces.

Lemma 5.5. *Every valid linear ordering satisfies the partial ordering $\mathcal{P}$.*

Proof. Consider a pair of adjacent faces u and v in $\mathcal{P}$ such that $u < v$. We constructed $\mathcal{P}$ such that $u < v$ if u precedes v, which by Lemma means that u comes before v in every valid linear ordering. Therefore for all $u < v$ in $\mathcal{P}$, u comes before v in every valid linear ordering. This means the linear ordering satisfied $\mathcal{P}$. $\square$

We will now introduce a concept which will be pivotal to our discussion of valid linear ordering. A *butterfly* B in a map $\mathcal{M}$ is a pair of adjacent faces u and v, where u precedes v, together with their common edge e. We denote this butterfly by $B = (u, v, e)$. We call u and v the *wings* of B and e the *hinge* of B. Two butterflies B_1 and B_2 are said to be *twin butterflies* in a particular flat folding of $\mathcal{M}$ if their hinges lie above the same edge of the unit square in some flat folding of $\mathcal{M}$.

Lemma 5.6. *"B_1 and B_2 are twin butterflies" is an equivalence relation on the set of butterflies with four equivalence classes.*

Proof. Clearly "B_1 and B_2 are twin butterflies" partitions the set of butterflies, since the edge of each butterfly must lie above some edge of the unit square, and can lie above only one such edge. Since there are four edges of the unit square, there are four possible equivalence classes. $\square$

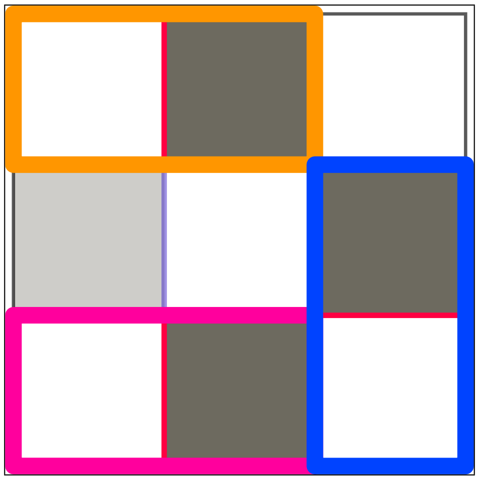


Figure 5.7: A 3×3 map with three butterflies highlighted.

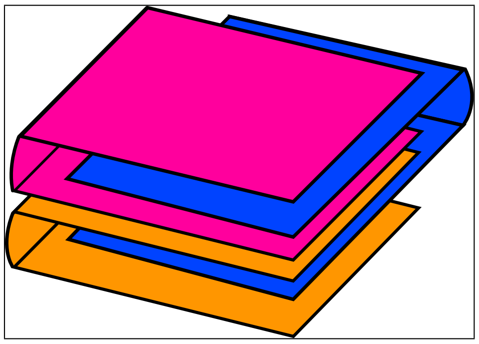


Figure 5.8: A subsection of the flat folded form of the map on the left. The orange and purple butterflies are twins, the blue butterfly is twins with neither of them.

We now explore the ways in which twin butterflies interact in a flat folded form. Suppose we have twin butterflies $B_1 = (u_1, v_1, e_1)$ and $B_2 = (u_2, v_2, e_2)$ We say that B_1 and B_2 *stack* in a particular linear ordering of faces if either (i) $u_1 < v_1 < u_2 < v_2$ or (ii) $u_2 < v_2 < u_1 < v_1$. We say that B_1 and B_2 *nest* in a particular linear ordering of faces if either (i) $u_1 < u_2 < v_2 < v_1$ or (ii) $u_2 < u_1 < v_1 < v_2$.

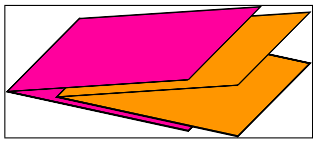


Figure 5.9: A pair of butterflies nesting.

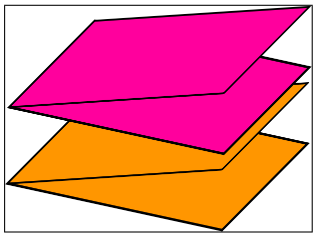


Figure 5.10: A pair of butterflies stacking.

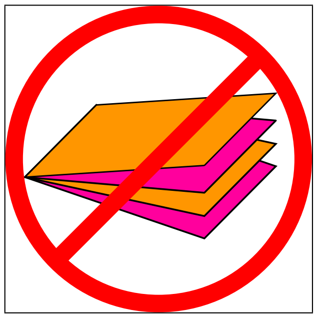


Figure 5.11: An example of an unallowed configuration of butterflies.

We say a linear ordering $\mathcal{L}$ of faces of an $m \times n$ map $\mathcal{M}$ satisfies the *butterfly condition* if every pair of twin butterflies in $\mathcal{M}$ either nest or stack in $\mathcal{L}$. It turns out that every linear ordering must satisfy this condition in order to be flat-foldable.

Lemma 5.7. *Every valid linear ordering satisfies the butterfly condition.*

Proof. Suppose for the sake of contradiction that we have a valid linear ordering $\mathcal{L}$ for which butterfly condition is not satisfied. Then there exists some pair of twin butterflies $B_1 = (u_1, v_1, e_1)$ and $B_2 = (u_2, v_2, e_2)$ which does not either stack or nest in $\mathcal{L}$. Then the order of the faces of these butterflies must be either $u_1 < u_2 < v_1 < v_2$ or $u_2 < u_1 < v_2 < v_1$. In order for the faces to not intersect (in either case), the edges of butterflies must overlap. See fig. 5.11. $\square$

Taken together, the two conditions presented in Lemmas 5.5 and 5.7 are sufficient for determining whether a linear ordering is valid.

Theorem 5.8. *A linear ordering $\mathcal{L}$ is valid if and only if (i) $\mathcal{L}$ satisfies the partial ordering $\mathcal{P}$ and (ii) $\mathcal{L}$ satisfies the butterfly condition.*

Proof. Suppose $\mathcal{L}$ is valid. Then we know by Lemma 5.5 that our valid $\mathcal{L}$ satisfies (i) the partial ordering $\mathcal{P}$. We also know by by Lemma 5.7, that our valid $\mathcal{L}$ satisfies the butterfly condition.

Suppose we have a linear ordering $\mathcal{L}$ of faces which satisfy (i) and (ii). Consider stacking the faces of the map on top of the unit square in the order determined by $\mathcal{L}$ where a face is dark-side up in our stack if it is dark-side up in the checkerboard pattern, and light-side up otherwise.

Consider some pair of faces in our stack which are adjacent in our map, and therefore are the wings of some butterfly. We can these faces together along the edge e of the butterfly. Since by (i) our linear ordering $\mathcal{L}$ satisfies the partial ordering $\mathcal{P}$, we know that the crease type made at e is the assignment of crease between u and v in our map, and not the opposite. (If $\mathcal{L}$ did not satisfy the partial ordering, we might have u before v or v before u, and thus the crease might be either a mountain or a valley.)

We can use this method to fuse together every pair of adjacent faces in our map. This will not create intersections between twin butterflies by (ii), since by the butterfly condition every pair of butterflies stacks or nests in $\mathcal{L}$, and stacking or nesting butterflies obviously do not intersect one another. Additionally, if a pair of butterflies are not twins, then they cannot possibly intersect. Therefore we have fused together every pair of adjacent faces in our map without creating any intersections. Since we have fused our map back together this is a folded form of the map, and since we have no intersections it is a flat-folded form. $\mathcal{L}$ is the linear ordering of the faces of this flat-folded form, and is therefore valid.

$\square$

We now use this result to develop an efficient algorithm for determining the validity of a given linear ordering of faces.

Corollary 5.9. *The validity of a linear ordering of the faces of a map $\mathcal{M}$ can be tested in polynomial time.*

Proof. There are $\mathcal{O}(m^2n^2)$ pairs of faces, therefore fewer than $\mathcal{O}(m^2n^2)$ pairs of *adjacent* faces, meaning there are $\mathcal{O}(m^2n^2)$ butterflies. (Note: if $f(n) < g(n)$, then $f(n)$ is in $\mathcal{O}(g(n))$.) Therefore there are at most $\mathcal{O}(m^4n^4)$ possible pairs of butteflies. For each butterfly B, we must check that the faces satisfy the partial order $\mathcal{P}$. This can be done in constant time for each of the $\mathcal{O}(m^2n^2)$ butterflies. We must also check, for every twin of B, whether they stack or nest. This can also be tested in constant time for each of the $\mathcal{O}(m^4n^4)$ possible pairs of butterflies. Therefore this algorithm runs in $\mathcal{O}(m^2n^2) + \mathcal{O}(m^4n^4) = \mathcal{O}(m^4n^4)$ time, which is $\mathcal{O}((mn)^4)$. Since mn is the size of our input, this is polynomial in the size of our input. $\square$

5.3 Complexity of Map Folding

In this section, we will discuss some results related to the open problem of determining the complexity of Map Flat Foldability. There has been some work done on easier versions of the problem, where one of the dimensions of the map is fixed. As discussed in Section 5.1, the $1 \times n$ Map Flat Foldability problem is equivalent to the Stamp Flat Foldability problem, which can be solved in constant time. In [6], Demaine, Liu, and Morgan develop a $O(n^9)$ algorithm for the $2 \times n$ Map Flat Foldability.

We will now present an original theorem on the complexity of Map Flat Foldability based on the results of the previous section.

Theorem 5.10. Map Flat Foldability $\in$ NP

Proof. In order to prove that a problem is in NP, we must show that any proposed solution to a particular instance of the problem can have its correctness verified in polynomial time. A solution to Map Flat Foldability consists of a linear ordering of the faces of the map. By Lemma 5.9 we can determine whether a linear ordering is valid in polynomial time. Recall that a linear ordering is valid if and only if the map is flat-folded. This means that we can determine whether the linear ordering corresponds to a flat folding of the map in polynomial time, and therefore Map Flat Foldability $\in$ NP. $\square$

We will now develop an alternative way to represent maps as graphs that encodes information from both the partial order $\mathcal{P}$ and the concept of twin butterflies. The *induced graph* of a map $\mathcal{M}$ is a digraph G with a node for every face in $\mathcal{M}$ and a directed edge (u, v) from face u to face v if and only if u precedes v, along with a coloring of the edges such that two edges $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ have the same color if and only if their corresponding butterflies $B_1 = (u_1, v_1, e_1)$ and $B_2 = (u_2, v_2, e_2)$ are twins. (We say a colored digraph G is *inducible* if there exists some $m \times n$ map $\mathcal{M}$ with induced graph G.) We will prove that the question of MAP FLAT FOLDABILITY is equivalent to asking whether this graph can be embedded in a particular way. Before we do so, we must discuss a few concepts from graph theory.

We will be considering various embeddings of graphs in order to develop the terminology we need to discuss map folding as a graph theory problem. A *planar embedding* of a graph G is an embedding of the vertices and edges of G in the plane such that no edges of G cross. The complexity of determining whether or not a graph has a planar embedding is linear in the number of vertices of the graph [20].

We can also impose additional restrictions on our embedding if we want to. An *upwards planar embedding* of a directed graph G is a planar embedding of G where every edge is a curve with increasing y coordinates [20]. The problem of determining whether a graph G has an upwards planar embedding is NP-complete [9].

We now consider a particular 3-dimensional embedding of a graph that will be essential to stating the question we wish to ask about inducible graphs. A *k-page book embedding* of a graph G is an embedding of G in k half-planes (called *pages*) that meet at a single line (called the *spine*) such that all of the vertices lie on the spine, every edge is completely contained in exactly one page, and each page in planar. Yannakakis, in [24], proved that every graph has a 4-page book embedding.

In a k-page book embedding, we are generally allowed to put our edges in whichever page we would like. This next type of embedding restricts us to putting each edge in a particular page. A *k-page partitioned book embedding* of a colored graph G is a k-page book embedding of G such that each page contains edges of exactly one color. Determining whether a graph has a 2-page partitioned book embedding takes linear time [10]. Determining whether a graph has k-page partitioned book embedding is NP-complete [3]. Sadly, both proofs are far beyond the scope of this paper.

We can also put restrictions on the structure of the embeddings in each page. A *k-page upwards book embedding* of a directed graph G is a k-page book embedding where every page is upwards planar. There has been almost no work done on the topic of k-page upwards book embeddings.

Now that we have a way of talking about these embeddings, we *finally* have the structure we need, and we can make our claim about map folding being equivalent to determining whether or not the induced graph has a particular type of embedding. (Both this claim and the "proof" are original work.)

Lemma 5.11. *A map $\mathcal{M}$ is flat foldable if and only if the butterfly coloring of the induced graph G has a 4-page upwards partitioned book embedding with page assignments given by coloring of G.*

Proof. A rigorous proof of this claim is beyond the scope of this paper, but we will give the proof idea here. The linear ordering of faces corresponds with the vetices/spine of the book, and the butterflies correspond to the edges. There are 4 pages, one for each of the equivalence classes of twin butterflies. A page is planar if and only if the butterflies correspondings to that equivalence class all stack or nest in the linear ordering. A page is upwards if and only if the partial ordering $\mathcal{P}$ is satisfied by our linear ordering. These two taken together say that all pages are upwards planar if and only if the linear ordering satisfies the butterfly condition and the partial ordering $\mathcal{P}$, which was exactly the condition for being a valid linear ordering according to Theorem 5.8. Therefore our linear ordering is valid if and only if our embedding is a 4-page upwards book embedding. But a valid linear ordering means a flat folded form, so we have a 4-page upwards partitioned book embedding (with page assignments given by the butterfly coloring of G) if and only if our form is flat folded. This means that a flat folding of $\mathcal{M}$ is possible if and only if the butterfly coloring of the induced graph G has a 4-page upwards partitioned book embedding with page assignments given by coloring of G. $\square$

This connection will allows us to use future results about 4-page upwards partitioned book embeddings to help determine the complexity of Map Flat Foldability. If we can find a polynomial time algorithm for determining if a graph has a 4-page upwards partitioned book embedding, then Map Flat Foldability is in P. Similarly, if we can prove that determining if an *inducible* graph has a 4-page upwards partitioned book embedding is NP-hard (as opposed to proving it for a general graph), then we would know that Map Flat Foldability is NP-complete. As it is, we have put down the mortar for future research to lay its bricks upon.

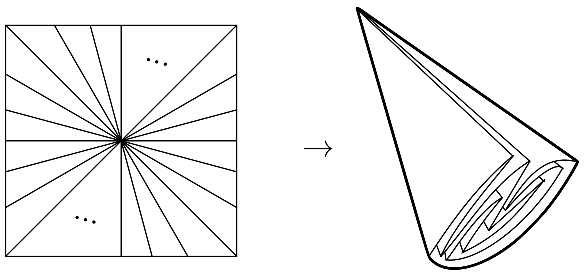


Figure 6.1: A star crease pattern with a sample flat folding.

6 The Combinatorics of Flat Folding

There are a number of countings problems we might ask vis-a-vis maps:

- How many folded forms are there for an $m \times n$ map?
- How many $m \times n$ map crease patterns admit flat foldings?

Both of these problems are very difficult; in fact so far they've been found intractible. Attempts at simplification have also yielded very few results: restricting the first question to the case of $1 \times n$ maps still produces a very difficult problem, and doing so to the second question yields a trivial answer: all $1 \times n$ crease patterns can be folded [4]. The question of how many $2 \times n$ map crease patterns admit flat foldings is again intractible.

Thus we look to another class of crease patterns for a manageable counting problem. As pictured in fig. 6.1, suppose P is an unassigned crease pattern on a circular paper with one vertex in the center and $2n$ creases space equiangularly about the vertex. We call P an *star* crease pattern. If we use $\mathbf{M}_n$ to denote the set of all possible flat-folded forms of this crease pattern, what is $|\mathbf{M}_n|$?

Perhaps unsurprisingly, the answer to this question is that no one is really sure. However, we do have a name for these objects: they are closed meanders. Indeed, as the canonical combinatorial folding problem, the single-vertex crease pattern question is one statement of a combinatorial class whose enumeration has escaped mathematicians for quite some time. Closed meander enumeration is a member of a group of problems in combinatorics that are what we might call 'elementary but hard'—no special mathematics is required to understand the statement of the problem, but solutions are still hard-won.

We begin with a formal definition of the problem.

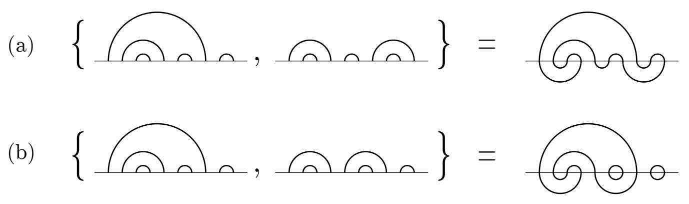


Figure 6.2: (a) An example of a closed meander of order 5, and (b) a pair of arch systems that is *not* a meander.

6.1 Definition

Definition 6.1 (Arch System)**.** *An arch system of order n is a collection of nested semicircles placed along a line such that no two arches intersect and the arch 'feet' are equidistant.*

Definition 6.2 (Closed Meander)**.** *A closed meander of order n is a pair of arch systems of order n (A,B) such that when B is reflected and placed on the same line and upside-down relative to A, the two arch systems form a single closed curve.*

As with many of definitions that follow, closed meanders are best understood pictorally; see fig. 6.2 for an example. We will use 'meander curve' to refer to the closed curve created by A and B, and 'meander line' to refer to the horizontal line in the (A, B). $\mathbf{M}_n$ will denote the set of meanders of order n and $M_n = |\mathbf{M}_n|$ will denote the number of meanders of order n. Hence we seek an efficient enumeration method for M_n.

A couple notes are in order: first, by a straightforward construction provided in [15], given an arch system A we can always find an arch system B such that (A, B) is a meander. After a little thought we realize that the set of arch systems of order n is in bijection with parenthesis sequences: just "shave off" the top portions of each arch to obtain a pair of parentheses. Because parenthesis sequences are counted by the Catalan numbers C_n, this gives us a lower bound on M_n:

$$M_n \geq C_n = \frac{1}{n+1}\binom{2n}{n}.$$

Secondly (and critically) not every pair of arch systems (A, B) yields a meander. This is because when A and B are placed opposite to each other, they might create multiple closed curves (see fig. 6.2(b)). Thus C_n^2, the number of pairs of arch systems of order n, is only a loose upper bound for M_n. In summary we have $C_n \leq M_n \leq C_n^2$; and referring to Table 6.3 for the

n	C_n	M_n	C_n^2
1	1	1	1
2	2	2	4
3	5	8	25
4	14	42	196
5	42	262	17,64
6	132	1,828	17,424
7	429	13,820	184,041
8	1,430	110,954	2,044,900
9	4,862	933,458	23,639,044
10	16,796	8,152,860	282,105,616

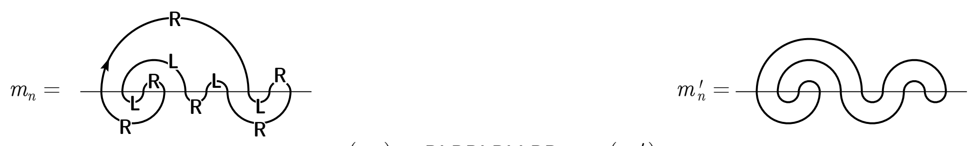
Figure 6.3: The first 10 terms of M_n and its Catalan number bounds.

beginning of each sequence, we see these bounds are (at least initially) very loose.

It is in this second fact that the difficulty of meander enumeration lies: how do we determine when a pair of arch systems produces a single closed curve? The naive answer is an $\mathcal{O}(n)$ algorithm—hardly conducive to the development of a closed form enumeration or recursive definition. Most extant work circumvents this problem by defining and studying a larger class of *meandric systems* $\mathbf{M}_n^k$ where k is the number of closed curves that the two arch systems produce. In this conception, enumerating closed meanders becomes determining M_n^1. While existing work has not solved the closed meander problem, the investigation of closed meanders within this larger set of objects has led to the discovery of many connections with other well-studied objects. One such connection is a natural bijection between meandric systems and reduced members of the Temperley-Lieb algebra, an algebra from statistical mechanics used to study knots and the braid group [8].

In what follows we take a different (and to the best of our knowledge, original) approach. We do not consider $\mathbf{M}_n^k$; instead we develop a recursive generation method for $\mathbf{M}_n$ while demanding, at each step, that we are working with a single closed curve. While we do not solve the closed meander problem, we will see that this recursive generation technique allows us to count an important subclass of meanders, in the end allowing us to reduce the general closed meander problem to a sensible subproblem.

To this end, we begin by defining a word-like structure on meanders that has the property of 'single-curve-connectness' built in. We then define raising "insertion" and lowering "deletion" operators on this structure to



$\pi(m_n) = \mathsf{RLRRLRLLRR} = \pi(m'_n)$

Figure 6.4: On the left we determine the winding sequence for the meander used in fig. 6.2. Notice that this π is the same as for the meander m'_n on the right.

organize meanders into a graded poset.

6.2 Winding Sequences

Definition 6.3 (Winding Sequence)**.** *The winding sequence $\pi(m_n)$ for a meander $m_n = (A, B)$ of order n is a $2n$-symbol word obtained from the alphabet $\{\mathsf{R}, \mathsf{L}\}$ in the following way:*

1. *Assign an orientation to the meander curve created such that the leftmost arch in A is oriented clockwise.*
2. *Beginning at the leftmost intersection with the line, append an* R *to π if the curve bends to the right relative to the orientation, and an* L *if the curve bends left.*
3. *Move along the curve in the direction of the assigned orientation until another intersection is reached. Repeat step* 2.
4. *Repeat this until we reach the beginning of the meander.*

In this description we understand each arch a_i as either an L arch or an R arch, writing $a_i = \mathsf{L}$ or $a_i = \mathsf{R}$, depending on what the i^{th} symbol is in the meander's winding sequence. For example, see fig. 6.4.

The image of π (the set of winding sequences that have an associated meander) are actually simple to describe: $\pi(m_n) =$ one R followed by an equal number of Ls and Rs and then a final R. We save the proof of this fact for the end of this section, by which point we will have developed sufficient machinery. Note that $\pi(m_n)$ does not uniquely determine m_n: while by the constructive nature of the definition every meander has a winding sequence, it is often the case that a given π is the winding sequence for multiple meanders, as demonstrated by fig. 6.4.

Given some meander m_n, one systematic way to obtain a new meander m'_n with the same winding sequence as m_n is by "shuffling" arches in the following way:

Definition 6.4 (Arch Exchange)**.** *For this definition we represent arches in an arch system as ordered pairs of 'arch feet'* $a = (a_1, a_2)$ *where* a_i *is equal to the index along the meander line of that given foot. Now suppose in a meander* (A, B) *we have two arches* $a = (a_1, a_2) \in A$ *and* $b = (b_1, b_2) \in B$ *where* $b_1 = a_2 + 1$*. Then map the index of each arch foot* c_i *in each arch* $c \in A \cup B$*, by*

$$c_i \to \begin{cases} c_i - b_1 + a_1 & b_1 \le c_i \le b_2 \\ b_2 - b_1 + c_i & a_1 \le c_i \le a_2 \\ c_i & \textit{otherwise.} \end{cases}$$

Informally an arch exchange swaps two arches by sliding them past each other while keeping the rest of the meander the same. See fig. 6.5 for an example. We appeal to this physical intuition to justify two facts: first, that arch exchange does not produce any arch crossings and thus yields a valid meander, and second, that if a meander m_n undergoes an arch exchange to become a different meander m'_n, its winding sequence is unchanged. In symbols, $\pi(m) = \pi(m')$.

We are now prepared introduce the fundamental "unit" of our meander generation process.

Definition 6.5 (Arch Pair)**.** *An* arch pair *is a pair of consecutive arches* $a_i a_{i+1} \in a_2 \cdots a_{2n-1}$ *in a winding sequence with* $a_i \neq a_{i+1}$*.*

Lemma 6.6. *Every meander of order 2 or greater contains an arch pair.*

Proof. Suppose it does not. Then either all $a_i \in a_2 \cdots a_{2n-1}$ are R or all $a_i \in a_2 \cdots a_{2n-1}$ are L. This is false. □

The constant presence of arch pairs in (≥ 2)-meanders suggests that a consistent process for arch pair deletion would provide a good method

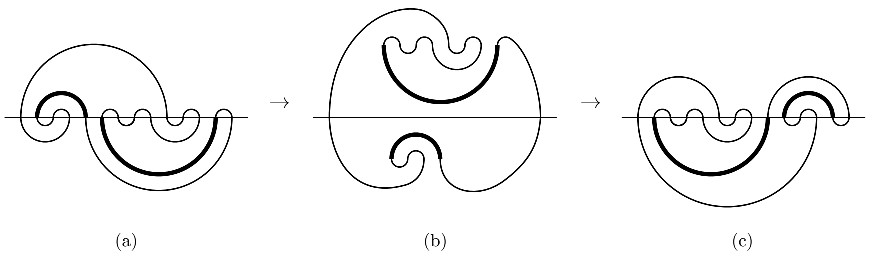


Figure 6.5: An intuitive representation of the arch exchange process. We exchange the bolded arches in the pictured meander by "stretching" the groups of arches around each other.

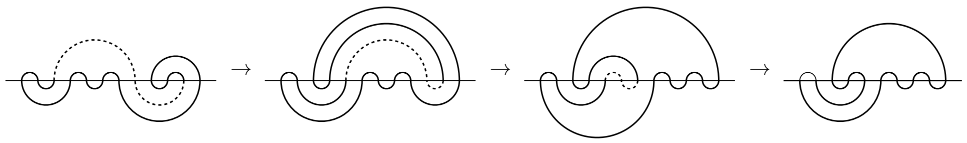


Figure 6.6: Arch pair deletion. We delete the dashed arch pair by swapping arches out of the right arch of the arch pair, swapping arches out of the left of the arch pair, then finally removing the arch pair. Note of course that (b) is a not a legitimate meander.

to obtain meanders of order $n-1$ from those of order n. Arch exchange provides us with just such a method.

Definition 6.7 (Arch Pair Deletion)**.** *If $a_i a_{i+1}$ is an arch pair in m_n, the deletion operator $D(m_n, i)$ removes it in the following way.*

1. *Exchange the arch within a_{i+1} that is closest to the arch a_i with a_i. Repeat this process until there are no arches within a_{i+1}.*
2. *Exchange the arch within a_i that is closest to the arch a_{i+1} with arch a_{i+1}. Repeat this process until there are no arches within a_{i+1}.*
3. *Delete arches a_i and a_{i+1} and connect a_{i-1} to a_{i+2}. (Or, equivalently, shrink $a_i a_{i+1}$ down to a point.)*

The deletion operator removes one arch from each arch system and keeps the meandric curve closed, so D maps meanders of order n to those of order $n-1$. Further, because arch exchange preserves winding sequence, deletion also preserves the portion of the winding sequence not deleted: *i.e.,* if $\pi(m_n) = a_1 \cdots a_{2n}$, then $\pi(D(m_n, i)) = a_1 \cdots a_{i-1} a_{i+2} \cdots a_{2n}$. See fig. 6.6 for an example. Because every meander of order $n \geq 2$ contains an arch pair, we may repeatedly remove arch pairs from an m_n to eventually obtain the 1-meander (with winding sequence RR).

This means we can use this arch pair deletion operator as a covering relation to obtain a poset of meanders: we say m covers m' if we can delete an arch pair in m to obtain m'. Further, if we use the order of the meander as a rank function, this poset is graded: if $m > m'$, then there is a series of deletions taking us from m to m' and so the order of m must be greater than m'; also, because deletion reduces order by one, then m covering m' implies the rank of m is 1 greater than the rank of m'.

However, if we want to generate $M = \bigcup_{n=1}^{\infty} M_n$ recursively, we need to be able to move up the poset rather than down. By extension of the deletion (lowering) operator, we might expect the existence of inverse operation (a raising operator) that inserts an arch pair to build a meander of order n from one of order $n-1$:

Definition 6.8 (Arch Pair Insertion)**.** *The insertion (or 'splicing') operator $S(m_n, p, i, j, k)$ inserts an arch pair $p = a_p a'_p \in \{\mathsf{RL}, \mathsf{LR}\}$ between arches a_i and $a_{i+1}, i \in \{1, \cdots, 2n-1\}$ in meander $m_n \in M_n$ in the following way.*

1. *Locally insert $a_p a'_p$ between arches a_i and a_{i+1}.*
2. *Exchange the j arches closest to a_p with a_p so that they now lie within a'_p.*
3. *Exchange the k arches closest to a'_p with a'_p so that they now lie within a_p.*

It should be noted immediately that S is not very well behaved. In particular S is not defined over all of $M_n \times \{\mathsf{LR}, \mathsf{RL}\} \times \mathbb{N}^3$: the number of arches available for exchanging with a_p and a'_p depend on the particular m_n receiving the insertion. However, note that we can of course always perform $S(m_n, p, i, 0, 0)$ on a given meander for suitable i. This last observation allows us to prove the form of winding sequences of meanders:

Theorem 6.9. *The set of winding sequences of meanders of order n is*

$$W_n = \left\{ \mathsf{R}x\mathsf{R} : x \textit{ is a permutation of } n-1 \ \mathsf{R}s \textit{ and } n-1 \ \mathsf{L}s \right\}.$$

Proof. Let $\pi = a_1 \cdots a_{2n}$ be a winding sequence for a meander m_n. Then π contains an arch pair $a_i a_{i+1}$. Delete it, yielding a meander m_{n-1} with the winding sequence $\pi' = a_1 \cdots a_{i-1} a_{i+2} \cdots a_{2n}$. Because every meander has an arch pair, we can repeat this process to eventually obtain the 1-meander with $\pi_1 = \mathsf{RR}$. Because we removed Ls and Rs in pairs, there were an equal number of them to start with in the substring $a_2 \cdots a_{2n-1}$.

Given a winding sequence π of the form above, we now construct a meander m_n with $\pi(m_n) = \pi$. To do this, create a list of winding sequences $\pi_n, \pi_{n-1}, \ldots, \pi_2, \pi_1 = \mathsf{RR}$ by deleting an arch pair from π_k to create π_{k-1}. Now construct a meander with winding sequence π by starting with the 1-meander and applying $S(m_k, p_k, i, 0, 0)$ on the k^{th} meander, where $p_k \in \{\mathsf{RL}, \mathsf{LR}\}$ is the arch pair removed at index k at the $(n-k)^{\text{th}}$ step in our sequence list. $\square$

As mentioned before, the available insertions at a particular point in a given meander is highly unpredictable: if we have some unknown meander m_n and wish to insert an arch pair into position i, it is unclear how many

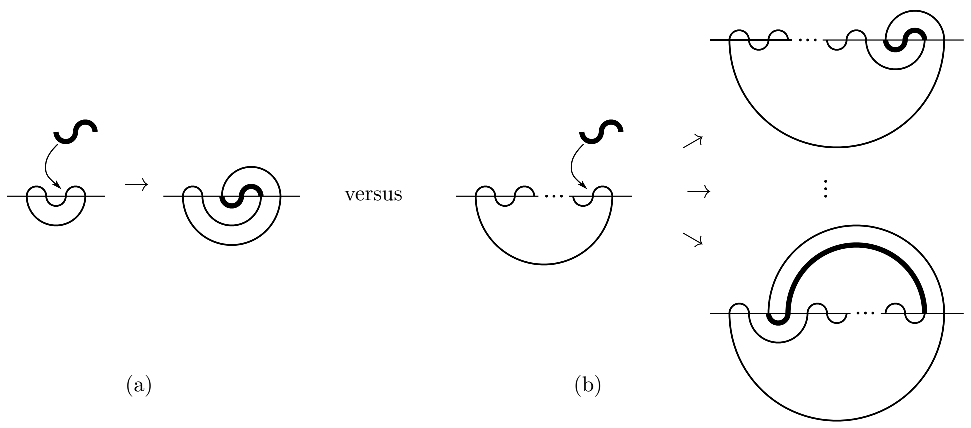


Figure 6.7: Some arch insertions may allow no arch exchanges (a) or an arbitrary number (b), depending on the meander receiving the insertion.

arch exchanges we are allowed to do, if any. Indeed, there may be insertions where there are no arches available for exchanging (fig. 6.7(a), because exchanging the first arch would change the winding sequence) or insertions where we there is an arbitrary number of exchanges possible (fig. 6.7(b)).

Fortunately, the following theorem partially alleviates our concern by greatly decreasing the number of arch exchanges we must consider during arch insertion. In particular, it states that we can generate any meander from RR by repeated insertions where we make one or zero arch exchanges at each insertion step.

Intuitively, $h(a_i)$ is the number of arches you would see if you stood on the edge of (a_i) and looked inside. For instance, in fig. 6.2, the first arch in the winding sequence has height 2.

Theorem 6.10 (At Most One Exchange)**.** *All meanders can be generated from* $m_1 = \mathsf{RR}$ *with repeated insertions* $S(m_n, p, i, j, k)$*, where* $p \in \{\mathsf{LR}, \mathsf{RL}\}$*,* $i \in \{2, 3, \cdots, 2n-1\}$ *and* $0 \leq j + k \leq 1$*.*

Before we present a proof we require one final definition. Its importance in the proofs of many of the results that follow makes it deserving of its own label.

Definition 6.11 (Arch Height)**.** *Given an arch* a_i *in* m_n*, we say its* arch height $h(a_i)$ *is the number of arches directly inside* a_i*.*

Now the proof.

Proof. The theorem is vacuously true for $n = 1$, so let $m_n \in M_n$ with $n \geq 2$.

Using exponents on Ls and Rs to denote arch height, we will prove this result by demonstrating that every meander contains an arch pair in this set:

$$P = \{\mathsf{R}^0\mathsf{L}^0,\ \mathsf{L}^0\mathsf{R}^0,\ \mathsf{R}^1\mathsf{L}^0,\ \mathsf{R}^0\mathsf{L}^1,\ \mathsf{L}^1\mathsf{R}^0,\ \mathsf{L}^0\mathsf{R}^1\}.$$

This will prove our result because each arch exchange in the insertion process increases the arch height by one.

Now define the *total height* of m_n to be

$$\mathcal{H}(m_n) = \sum_{a \in m_n} h(a).$$

The proof proceeds in two parts. The first establishes a (sharp) upper bound on $\mathcal{H}(m_n)$. The second demonstrates that a meander containing none of the arch pairs in our generating set P violates this upper bound.

To bound $\mathcal{H}(m_n)$, we first transform m_n into a pair of arch systems (not necessarily a meander) that is easier to count but preserves total height. This *chaining* transformation $C : \mathbf{M}_n \to \mathbf{M}_n^*$ is defined by the following algorithm: if there is an arch $a \in m_n$ with at least two child arches b, c, expand b to encompass c. Repeat. Note that at each step the algorithm does not change $\mathcal{H}(m_n)$ because arch a loses a child arch but arch b gains one. Hence $\mathcal{H}(m_n) = \mathcal{H}(C(m_n))$. This process is demonstrated in fig. 6.8.

Now define an *outer arch* in m_n to be any arch which is not the descendant of any other arch. Let $N(m_n)$ be the number of outer arches in m_n and note that $N(m_n) = N(C(m_n))$ because only arches contained within other arches are modified by the chaining algorithm. We are now prepared to count $H(m_n)$. Observe that $C(m_n)$ is composed of 'chains' of nested arches, each with height 1 except for the last arch on the chain that has no children and thus height 0. There are $2n$ arches in total and one outer arch for each arch of height 0, so we have

$$\begin{aligned} \mathcal{H}(C(m_n)) &= 1 \cdot (2n - N(C(m_n))) + 0 \cdot N(C(m_n)), \\ \text{thus } \mathcal{H}(m_n) &= 2n - N(m_n). \end{aligned}$$

It is clear that every m_n has at least two outer arches, but if $N(m_n)$ has exactly two outer arches then they must be connected in a circle, implying that $n = 1$. Hence for $n \geq 2$, $N(m_n) \geq 3$, so

$$\mathcal{H}(m_n) \leq 2n - 3.$$

While not required for the proof, it is interesting to note that this bound is sharp because a 'spiral meander' of order n has three outer arches (see fig. 6.8(c) for an example of a spiral meander of order 4).

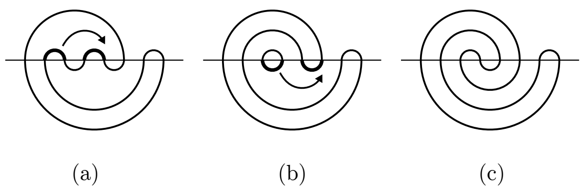


Figure 6.8: An example of the chaining operation. This meander requires two chaining steps, first on the two bolded arches in the upper arch system (a), and then on the two bolded arches in the lower arch system (b). The fully chained meander is pictured in (c).

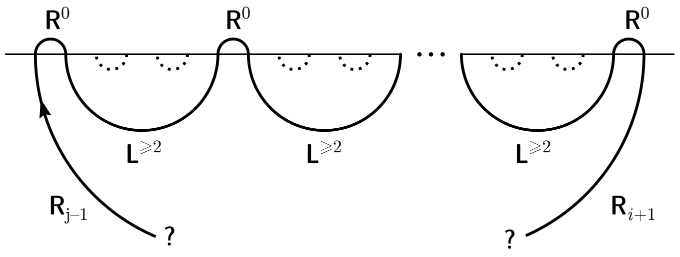
Now suppose there exists an m_n that does not contain any arch pairs from P. We determine a lower bound on $\mathcal{H}(m_n)$ by examining arches of height 0. If some 0-arch a^0 is in an arch pair, the other arch in the pair a' must have height $h(a') \geq 2$ because otherwise we would have an arch pair from P. Note that each arch a^0 must be in at least one arch pair because otherwise we would be forced to set $m_n = \mathsf{R}^0\mathsf{R}^0$ to avoid overlaps, and this is not allowed because we have assumed $n \geq 2$.

The following algorithm builds an injective function ϕ from the set of 0-arches A_0 in m_n to the set of arches $a \in m_n$ with height $h(a) \geq 2$, denoted A_2, by greedily associating arches along the winding sequence.

1. Move along the winding sequence until a 0-arch a_i^0 is found. As noted earlier, a_{i-1} or a_{i+1} forms an arch pair with a_i^0.
   (a) If a_{i-1} does not form an arch pair with a_i^0, a_{i+1} does. Set $\phi(a_i) = a_{i+1}$.
   (b) If a_{i-1} forms an arch pair with a_i^0 and is unassociated, set $\phi(a_i) = a_{i-1}$.
   (c) If a_{i-1} is already associated, then a_{i+1} forms an arch pair with a_i^0 by Lemma 6.12 below. Set $\phi(a_i) = a_{i+1}$.
2. Repeat step 1 until ϕ is defined for all 0-arches.

Let k be the number of arches in A_0 and let ℓ the number of arches in A_2. Because the above algorithm demonstrates an injection from 0-arches to $\geq$ 2-arches, $\ell > k$. All other $2n - k - \ell$ arches in m_n must be height ≥ 1, so we have

$$\begin{aligned}\mathcal{H}(m_n) &\geq 0k + 2\ell + 1(2n - k - \ell)\\ &\geq 0k + 2k + 2n - k - k\\ &= 2n,\end{aligned}$$



R^0
R^0
R^0
$\mathsf{L}^{\geqslant 2}$
$\mathsf{L}^{\geqslant 2}$
$\mathsf{L}^{\geqslant 2}$
$\mathsf{R}_{j\text{-}1}$
R_{i+1}
?
?

Figure 6.9: The impossible situation described in Lemma 6.12.

contradicts the upper bound found previously. $\square$

Lemma 6.12. *In the association algorithm step 1(c), if a_{i-1} is already associated, then a_{i+1} forms an insertable arch pair with a_i.*

Proof. Suppose not. Then $a_k = a_{k+1}$. Because a_{i-1} is already associated, somewhere to the left in the sequence there is a 0-arch a_j^0 that fits condition (a). Hence we have a subsequence (up to direction)

$$\cdots \mathsf{R}_{j-1}\mathsf{R}_j^0\mathsf{L}_{j+1}\mathsf{R}_{j+2}^0 \cdots \mathsf{L}_{i-1}\mathsf{R}_i^0\mathsf{R}_{i+1} \cdots .$$

Graphically, this situation looks like fig. 6.9. The outer ends of both R_{j-1} and R_{i+1} may not land within any of the $\mathsf{R}_j, \mathsf{R}_{j+2}, \cdots, \mathsf{R}_i$ because that would violate the fact that $\mathsf{R}_j, \mathsf{R}_{j+2}, \cdots, \mathsf{R}_i$ are 0-arches. Nor can they both land outside of the portion of the meander created by the subsequence, for that would create a crossing. Hence R_{j-1} and R_{i+1} must connect to each other; *i.e.*, the subsequence is the entire meander. But then there would be no arches under each L in the subsequence, contradicting the assumption that their height is ≥ 2. $\square$

This theorem implies that if we restrict our covering relation to deletions that only require a single arch exchange, our poset will remain connected. For the remainder of this note we will concern ourselves with the subproblem of counting the meanders of order n that may be generated without any exchanges. We will call these *simple* meanders.

6.3 Enumerating Simple Meanders

To provide a formal definition of simple meanders, we will begin by providing a description of a path up the poset.

Definition 6.13 (Insertion Sequence)**.** *An insertion sequence σ of order n is a sequence of of $n-1$ ordered pairs $(p_1, i_1), \ldots, (p_{n-1}, i_{n-1})$ where $p_k \in \{\mathsf{LR}, \mathsf{RL}\}$ and $1 \leq i_k \leq 2k-1$. We denote the set of all insertion sequences S_n.*

Intuitively an insertion sequence σ is a "blueprint" for constructing a meander. Using our definition of insertion from before, we can define a way to produce a meander from σ.

Definition 6.14 (Meander Derivation)**.** *If $\sigma = (p_1, i_1), (p_2, i_2), \ldots, (p_{n-1}, i_{n-1})$ is an insertion sequence, we write $\mu(\sigma)$ to denote*

$$S(S(\cdots S(S(m_1, p_1, i_1, 0, 0)p_2, i_2, 0, 0)\cdots)p_{n-2}, i_{n-2}, 0, 0)p_{n-1}, i_{n-1}, 0, 0).$$

We are now prepared to formally state the definition of simple meanders.

Definition 6.15 (Simple Meander)**.** *A meander $m_n \in M_n$ is* simple *if there exists an insertion sequence σ such that $\mu(\sigma) = m_n$. We denote the set of simple meanders of order n as $\mathbf{M}_n^s$ and write $M_n^s = |\mathbf{M}_n^s|$.*

While the definition of $\mathbf{M}_n^s$ suggests it might be only a small subclass of $\mathbf{M}_n$, we argue that $\mathbf{M}_n^s$ is by some measures fairly important: we will see at the end of this paper that the structure it provides may reduce the entire meander enumeration problem to that of counting meanders in which *every* arch pair had to include an exchange at insertion. In other words, we may avoid having to count directly those meanders which only require *some* arch exchanges.

With that motivation in mind, we now engage fully with the problem of counting M_n^s. As defined each member $m_n \in \mathbf{M}_n^s$ is the output of compositions of $n-1$ functions $S(\cdots)$. In this notation the structure of meander derivation is very difficult to discern, so we now develop an alternative description of insertion sequences.

Definition 6.16 (Insertion Tree)**.** *An* insertion tree *of order n is an ordered (plane) tree where*

- *All internal nodes have degree three*
- *There are $n-1$ internal nodes*
- *All internal nodes have a label from $\{\mathsf{LR}, \mathsf{RL}\}$*
- *All other nodes are labeled $\bullet$.*

We will use T_n to denote the set of all insertion trees of order n.

Because all internal nodes have three children, we will call these children the left child, middle child, and right child respectively (orienting the tree with its root at the bottom). We will also abuse tree terminology and use "leaves" to denote nodes that have only $\bullet$ children and use "rails" to denote $\bullet$ nodes. This naming convention has a graphical motivation that will become evident shortly.

Definition 6.17 (Insertion Tree Diagram)**.** *An* insertion tree diagram *of order n is an ordered pair (T, γ) where T is an insertion tree of order n and γ is a linear ordering on the* LR *and* RL *nodes in T that satisfies the partial ordering perscribed by T. We denote the set of insertion tree diagrams of order n as D_n*

Insertion diagrams are drawn with all rails at at the top of the diagram (we can do this because they by definition have no children) and the linear ordering is read from bottom to top. See the left side of fig. 6.10 for an example of an insertion tree diagram.

Observe that insertion tree diagrams "are" insertion sequences: in analogy to insertion of arch pairs in meander, we can build the tree diagram corresponding to an insertion sequence $\sigma = (p_1, i_1), \cdots (p_k, i_k), \cdots (p_{n-1}, i_{n-1})$ by attaching a node labeled p_k at the ${i_k}^{\text{th}}$ rail in the extant tree diagram for all k. Conversely we can obtain the insertion sequence associated with an insertion tree diagram by starting at the bottom of the diagram and appending the ordered pair (p, i) with p being the label of the current node and i being one more than the number of labels to the left of the current node in T. See fig. 6.10 for an example.

Hence just as we defined $\mu : S_n \to \mathbf{M}_n^s$, we may analogously define $\mu : D_n \to \mathbf{M}_n^s$. This allows us to interpret a insertion tree diagram D further. Intuitively, as we move up the diagram, the rails represent the 'spaces' between arches in the winding sequence of $\mu(D)$, or, equivalently, the intersections of the meander curve with the meander line. Hence the placement of a node on a rail is equivalent to the insertion of an arch pair between the two corresponding symbols in the winding sequence. This observation allows us to read the winding sequence of $\mu(D)$ directly off the diagram as pictured in fig. 6.11: starting at the top of the diagram, begin with the first R that comes from our initial m_1, then for each arch that follows, follow the spaces between the rails down to the node that created them and record the corresponding arch direction. Finally, add the last R that comes from m_1.

It turns out that for any given $T \in T_n$, $\mu(T, \gamma) = \mu(T, \gamma')$ for all linear extensions γ, γ'. In other words, linear extensions of T don't matter—the

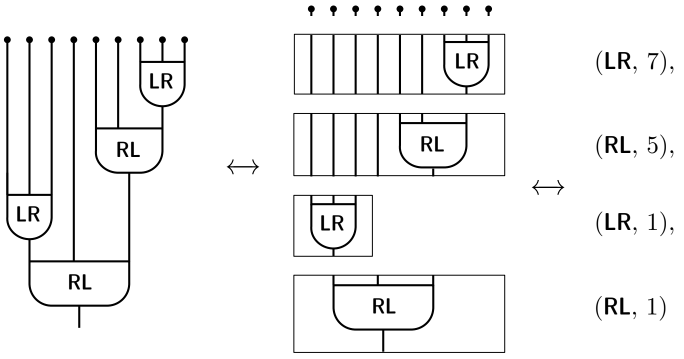


Figure 6.10: An example of the equivalence of insertion tree diagrams and insertion sequences for the sequence $\sigma = (\mathsf{RL}, 1), (\mathsf{LR}, 1), (\mathsf{RL}, 5), (\mathsf{LR}, 7)$.

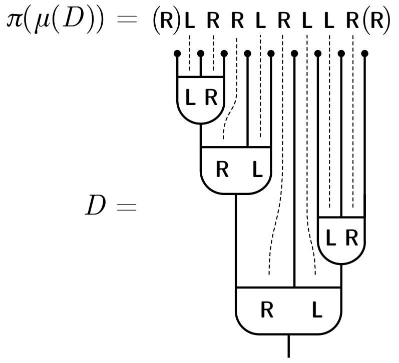


Figure 6.11: Reading the winding sequence from an insertion tree diagram D.

meander associated with an tree diagram depends only on the tree involved. We will not prove this fact for space reasons (indeed it will not be used in our counting argument), but it does motivate the search for a bijection between $\mathbf{M}_n^s$ and some subset of T_n, rather than some subset of D_n.

It is exactly this sort of bijection that we will prove. We begin by defining our subset of T_n, showing a bijection between this subset and a subset of S_n, and finally proving a bijection between this subset of S_n and the whole set $\mathbf{M}_n^s$.

Definition 6.18 (Left-greedy Insertion Trees). *A left-greedy insertion tree of order n is an insertion tree of order n where for each node v,*

– *if $v = \mathsf{RL}$, then the middle child of v is RL or $\bullet$ and the right child of v is LR or $\bullet$,*

– *if $v = \mathsf{LR}$, then the middle child of v is LR or $\bullet$ and the right child of v is RL or $\bullet$.*

We denote the set of these objects T_n^*.

This subset of insertion trees is in bijection with simple meanders. To show this, we first put T_n^* in bijection with a subset of S_n by way of the following invertible function:

Definition 6.19. *Define $\xi : T_n^* \to S_n$ as follows. Given a $T \in T_n^*$, assign a linear ording γ to its LR and RL nodes by starting with the empty linear ordering $\gamma = \cdot$ and then while T is nonempty, remove the leftmost leaf v from T, prepend it to γ, and repeat. We now have a insertion tree diagram (T, γ) from which we can write down the insertion sequence σ. This process is fully determined and we set $\xi(p) = \sigma$.*

It is clear ξ maps into S_n because our construction of γ obeys the partial ordering perscribed by T. Now define the set of *left-greedy insertion sequences* as $S_n^* = \operatorname{im} \xi \subseteq S_n$. We proceed by showing T_n^* is in bijection with S_n^*.

Lemma 6.20. *$\xi : T_n^* \to S_n^*$ is a bijection.*

Proof. By definition ξ restricted as $\xi : T_n^* \to S_n^* = \operatorname{im} \xi$ is onto. We now show that ξ is also 1-to-1 over this range. Suppose we have some $T_1, T_2 \in T_n^*$ such that $\xi(T_1) = \xi(T_2) = \sigma$. Now consider the insertion tree diagram D corresponding to σ. There is only one $D = (T, \gamma)$ by arguments made above. Hence $\xi(T_1)$ and $\xi(T_2)$ have the same tree diagram so $T_1 = T = T_2$. □

So now we have $|T_n^*| = |S_n^*|$. It remains to show $|S_n^*| = |M_n^s|$. To do this we will use the "meander production" function μ defined earlier as well as a new function δ that maps simple meanders back to elements of S_n^*. In order to define δ we first need a special name for arch pairs that can be deleted without shuffling: For any $m_n \in \mathbf{M}_n^s$, call an arch pair in m_n *deletable* if both arches involved have height 0.

Definition 6.21. *We define $\delta : \mathbf{M}_n^s \to S_n$ as follows. Begin with an empty insertion sequence $\sigma = \cdot$. Given a simple meander m_n, read along the meander in the order of its winding sequence until the first deletable arch pair $a_i a_{i+1}$ is encountered. Prepend $(a_i a_{i+1}, i)$ to σ and delete the arch pair from m_n to get a new meander of order m_{n-1}. Now repeat this process with the smaller meander and continue until $m_1 = \mathsf{RR}$ is reached. The resulting σ is the value of $\delta(m_n)$.*

The proof of this final bijection is lengthy, requiring multiple stages of induction arguments. We therefore split it into two lemmas, each of which is rather involved.

Lemma 6.22.

$$\operatorname{im} \delta \subseteq S_n^*.$$

Proof. We show that given an $m_n \in \mathbf{M}_n^s$, δ constructs an insertion tree diagram D that obeys the definition left-greedy trees, and it does so with a linear extension ordering that is left-greedy. We do this by describing the production of D step by step. Suppose $p_{n-1} = a_i a_{i+1}$ is the first arch pair removed by δ from m_n, where i is the index of the pair in $\pi(m_n)$. We can begin drawing D by placing $i-1$ rails to the left of a node labeled with the value of p_{n-1}. We do not yet know what is below or to the right of p_{n-1}.

Now consider the next arch pair $p_{n-2} = a_j a_{j-1}$ that δ removes from m_n. If $j > i$ then p_{n-2} is not a parent of any node already in D and its placement does not violate the definition of T^*. If $j \leq i$, then either p_{n-2} is the parent of rails (not a violation) or the parent of p_{n-1}. Now we much check that if p_{n-2} a parent of p_{n-1}, it is a valid node as perscribed by the definition T^*. Anything is allowed if p_{n-1} is the left child of p_{n-2}, so we address the other possibilities.

Suppose p_{n-1} is the middle child of p_{n-2}. For the sake of explicitness we declare $p_{n-2} = \mathsf{RL}$; because the definition of T^* is fully symmetric the argument will go through identically for $p_{n-2} = \mathsf{LR}$. By the definition of T^* and because we're assuming $p_{n-2} = \mathsf{RL}$ we therefore do not have a violation if $p_{n-1} = \mathsf{RL}$. Might it be the case that $p_{n-1} = \mathsf{LR}$? This would mean we

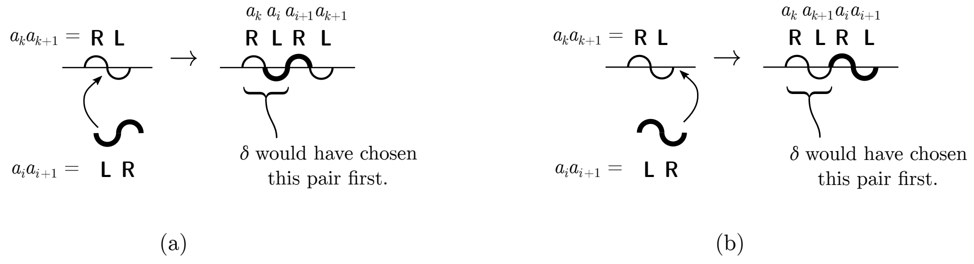

$a_k a_{k+1}$ = R L
$a_i a_{i+1}$ = L R
$a_k\ a_i\ a_{i+1} a_{k+1}$
R L R L
δ would have chosen this pair first.
(a)
$a_k a_{k+1}$ = R L
$a_i a_{i+1}$ = L R
$a_k\ a_{k+1} a_i a_{i+1}$
R L R L
δ would have chosen this pair first.
(b)

Figure 6.12: The two cases of contradiction involved in Lemma 6.22.

had in our original meander the subsequence $a_k a_i a_{i+1} a_{k+1} = \mathsf{RLRL}$. The middle $\mathsf{LR} = a_i a_{i+1}$ was deleted first, so $h(a_i) = h(a_{i+1}) = 0$ in the original meander, and the outer $\mathsf{RL} = a_k a_{k+1}$ was deleted second, so at that point $h(a_k) = h(a_{k+1}) = 0$ as well. But if we rewind this process we notice that inserting $\mathsf{LR} = a_i a_{i+1}$ in the middle of $\mathsf{RL} = a_k a_{k+1}$ does not increase the height of a_k or a_{k+1}. (See fig. 6.12(a)). Hence in the final meander we have $a_k a_i a_{i+1} a_{k+1} = \mathsf{R}^0\mathsf{L}^0\mathsf{R}^0\mathsf{L}^0$. This is a contradiction, because by the definition of δ, it would have deleted $a_k a_i$ first instead. Hence there is no violation.

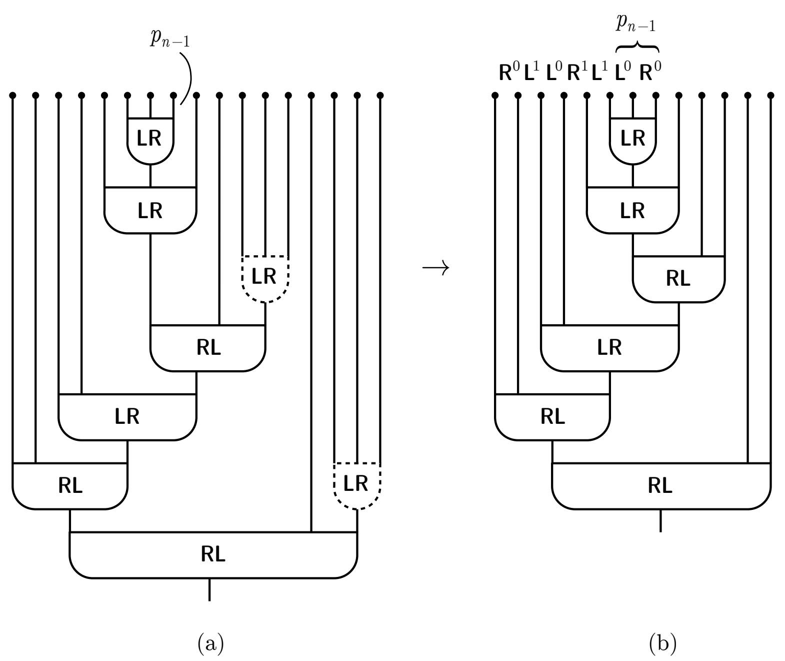
Now suppose p_{n-1} is the right child of $p_{n-2} = \mathsf{RL}$. If p_{n-1} is LR we're ok, so suppose instead that $p_{n-1} = \mathsf{RL}$. We are in a similar situation to the above, except we have $a_k a_{k+1} a_n a_{n+1} = \mathsf{RLRL}$. But, as pictured in fig. 6.12(b), inserting an RL pair directly after an RL pair does not increase the height of the first pair of arches, so we have a substring in our insertion sequence that looks like $\mathsf{R}^0\mathsf{L}^0\mathsf{R}^0\mathsf{L}^0$. This is a contradiction, because by the definition of δ, it would have deleted $a_k a_{k+1}$ first instead.

Hence there are no violations of the definition of a left-greedy insertion tree when we add p_{n-2} to our growing insertion tree diagram. The arguments above easily generalize to the rest of the arch pair insertions, and so at the end we are left with an insertion tree diagram (T, γ) where $T \in T^*$. It remains to check γ is a left-greedy linear extension of T^*. But this is clear from the way δ works. $\square$

We are almost prepared to show now prepared to show prove our bijection. We just need one more fact about the structure of elements of S_n^*.

Lemma 6.23. *If $\sigma \in S_n^*$, the last element in σ corresponds to the first deletable arch pair in $\pi(\mu(\sigma))$.*

Proof. Suppose the last element in σ is (p_{n-1}, i_{n-1}). Then the winding sequence for $\mu(\sigma)$ is $\mathsf{R}a_1 a_2 \ldots a_i a_{i+1} \ldots a_{2n-2}\mathsf{R}$ where $a_i a_{i+1} = p_{n-1}$. It suffices to show there is no deletable arch pair in $a_1 a_2 \ldots a_{i-1}$. Consider



p_{n-1}
LR
LR
LR
RL
LR
RL
LR
RL
$\rightarrow$
p_{n-1}
$\mathbf{R}^0\mathbf{L}^1\mathbf{L}^0\mathbf{R}^1\mathbf{L}^1\mathbf{L}^0\mathbf{R}^0$
LR
LR
RL
LR
RL
RL
(a)
(b)

Figure 6.13: An example of the arch height bounding processed using in 6.23. In (a) we remove nodes that are not in the path from the root to p_{n-1}. In (b) we label the arches $a_1 \dots a_{i-1}$ with lower bounds on their arch heights.

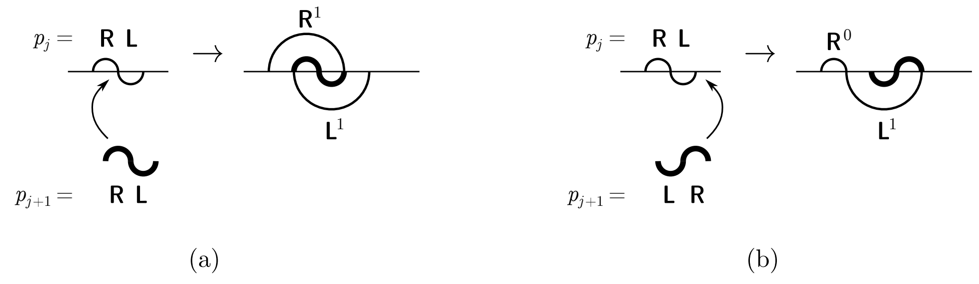


Figure 6.14: An illustration for the cases in Lemma 6.23: inserting arch pair p_{j+1} in p_j will increase the heights of arches in p_j as desired regardless of the type of p_{j+1}.

the insertion tree diagram $D = (T, \gamma)$ corresponding to σ. What nodes contribute an arch to the substring $a_1 \dots a_{i-1}$, and what is the height of each of these arches?

Because of the definition of ξ, $p_{n-1} = a_i a_{i-1}$ is certainly the leftmost leaf in T. Thus the nodes in T that contribute at least one arch to the substring in question are in the path from the root to p_{n-1}. We will examine this path separately, so define a new insertion tree diagram (T', γ') where T' is the path from the root of T to p_i and γ' is the subsequence of γ including the relevant nodes. For example, in fig. 6.13(a) we have an original T with the irrelevant nodes marked with dashes. They are then removed in T' as shown in fig. 6.13(b).

First note that the arch heights for $a_1 \dots a_{i-1}$ in $\pi(\mu(T', \gamma'))$ are lower bounds for those in $\pi(\mu(T, \gamma))$—the extra insertions in T (in comparison to T') certainly will not decrease the heights of $a_1 \dots a_{i-1}$. We now show that at least all the arches with odd index or all the arches with even index in $a_1 \dots a_{i-1}$ have height greater than or equal to 1, implying there is no deletable arch pair in this sequence.

We begin at the root of T' and move up, showing that the rightmost arch of each node's contribution to $a_1 \dots a_{i-1}$ has height 1 or greater. For this purpose we will call the root node p_1 and index the path $p_1, p_2, \dots, p_k = p_{n-1}$, proceeding by induction on p_j, $j = 1, 2, \dots k-1$. We will use $a_1 a_2 \dots a_\ell$ to keep track of the growing sequence of arches that are before $a_i a_{i+1}$ in the final arch sequence. Before we begin $\ell = 0$ and we have no deletable arch pairs in $a_1 \dots a_\ell$. Now assume there are no deletable arch pairs in $a_1 \dots a_\ell$ and consider the next node p_j in the path.

- If p_{j+1} is the left child of p_j, then p_j it does not add arches to $a_1 \dots a_\ell$ so it certainly does not create a deletable arch pair.

- If p_{j+1} is the middle child of p_j, then p_j and p_{j+1} are either both RL or LR. As shown in fig. 6.14(a), this means inserting p_{j+1} into p_j increases the height of both arches in p_j by one. Because p_j has a middle child, the left arch of p_j is added to $a_1 \dots a_\ell$ and because it has height greater than or equal to one it does not create a deletable arch pair.
- If p_{j+1} is the right child of p_j, then p_1 and p_{j+1} are opposite arch pairs (one is RL and the other is LR.). As shown in fig. 6.14(b), this means inserting p_{j+1} to the right of p_j increases the second arch in p_j by one. In arch height superscript notation, this means p_j appends $\mathsf{R}^0\mathsf{L}^{\geq 1}$ or $\mathsf{L}^0\mathsf{R}^{\geq 1}$ to $a_1 \dots a_\ell$. This will only create a deletable arch pair if the previous arch in $a_1 \dots a_\ell$ has height 0. But by this case and the previous case the rightmost arch in the sequence $a_1 \dots a_\ell$ always has height one. So the addition of p_j does not create a deletable arch pair.

Hence we do not create a deletable arch pair until the addition of p_{n-1} as desired. □

Theorem 6.24.
$$|T_n^*| = M_n^s$$

Proof. Lemma 6.20 tells us that $|T_n^*| = |S_n^*|$. It remains to show $|S_n^*| = M_n^s$.

First we claim function μ restricted as $\mu : S_n^* \to \mathbf{M}_n^s$ is onto. Suppose $m_n \in \mathbf{M}_n^s$. Then clearly the insertion sequence $\delta(m_n) = \sigma$ yields m_n when operated on by μ. By Lemma 6.22 $\sigma \in S_n^*$ and so we have an element σ in the domain of μ such that $\mu(\sigma) = m_n$. Because μ is onto we now have $|S_n^*| \geq M_n^s$.

We now show δ is onto, implying $|S_n^*| \leq M_n^s$ and thereby completing the proof. Suppose $\sigma = (p_1, i_1), \dots, (p_{n-1}, i_{n-1}) \in S_n^*$. Then we claim $\delta(\mu(\sigma) = \sigma$. We will prove this by induction on $j = n-1, n-2, \dots 2, 1$ as an index for elements in σ. By Lemma 6.23, the last element in σ corresponds to the first deletable arch pair in $\mu(\sigma)$, so this is exactly the first arch pair that δ prepends to $\mu(\sigma)$. The rest follows from induction on σ by repeatedly removing the last element in the sequence and applying Lemma 6.23 to show that this is exactly the next element δ prepends to $\delta(\mu(\sigma))$. □

It now remains to count $|T_n^*| = M_n^s$.

Theorem 6.25.
$$M_n^s = \begin{cases} 1 & \textit{if } n = 1 \\ 2H(n-1) & \textit{for } n \geq 2 \end{cases}$$

where $H(n) = 1$ *for* $n = 0, 1$ *and for* $n \geq 2$ $H(n)$ *is given by*

$$H(n) = 2\sum_{i=0}^{n-2}\sum_{j=0}^{n-i-2} H(i+1)H(j)H(n-2-i-j) + \sum_{i=0}^{n-1} H(i)H(n-1-i).$$

Proof. We count all the trees with n nodes (in T^*_{n+1}) with RL as a root and then mulitply by two. Thinking about constructing such a T from the root up in recursive fashion, consider having just laid a node v (*i.e.,* it is already set as LR or RL) with the promise to lay $n-1$ more nodes on v's subtree. We count by cases depending on the possible children of v.

If v has a left child that is not a rail, then the subtree of v's left child could have an RL root or an LR root. Because we understand v to be set, this means we must multiply by two in our recurrence to address the multiplicity in this case. However, the roots of the subtrees on the the middle and right children of v are predetermined by definition. Hence, for the case that v has a non-rail left child, the recurrence relation is twice the sum over of ways to distribute $n-1$ nodes among the subtrees of v's left child, middle child, and right child. Each summand is the product of the number of possible subtrees for each child because these are independent decisions. This is the first term in $H(n)$.

If v has a rail as a left child, we do not need to address multiplicity because the labels on the middle and right children of v are predetermined. Hence in this case the recurrence relation is the sum over of ways to distribute $n-1$ nodes among the subtrees of v's middle child and right child. Each summand is the product of the number of possible subtrees for these two children because these are independent decisions. This is the second term in $H(n)$. □

This completes our result for the enumeration of M^s_n. Table 6.15 displays this first 10 terms of the sequence in comparison to M_n. We see that the sequences are identical until $n = 5$; it is at this point that the first non-simple meander appears.

6.4 Further Study

The clear next step is to determine a closed form for M^s_n. Initial exploration into the OEIS using Table 6.15 suggests that M^s_n is exactly twice the sequence A003168, the "number of blobs with $2n+1$ edges." Many formulae

n	M_n^s	M_n
1	1	1
2	2	2
3	8	8
4	42	42
5	252	262
6	1,636	1,828
7	11,188	13,820
8	79,386	110,954
9	579,020	933,458
10	4,314,300	8,152,860

Figure 6.15: The first 10 terms of M_n^s versus M_n.

and generating functions are known for this sequence ([1]), including the elegant

$$\sum_{k=1}^{n} \frac{1}{n}\binom{n}{k}\binom{2n+k}{k-1}.$$

Beyond determining a direct solution to the recurrence relation for M_n^s, it may be fruitful to pursue bijective proofs with objects that have already been shown to be counted by sequence A003168.

How might we evaluate the determination of M_n^s as progress towards the enumeration of closed meanders? Perhaps predictably, we argue it is an important step in the right direction. Insertion trees provide a rich framework within which to study meanders; indeed we can place an algebra-like structure on $\mathcal{T} = \bigcup_{n=1}^{\infty} T_n$ by defining a way to add insertion trees onto rails of other insertion trees (in analogy to concatenating insertion sequences). We may also define commutation relations on $\mathcal{T}$ that give rise to equivalence classes of trees that yield the same meander.

However, the true power of insertion trees may lie in our ability to extend the set of allowable nodes to include *irreducible meanders* M_n^i, or meanders whose production requires a swap at *every* step. This would work by taking an irreducible meander, removing the first and last arch in its winding sequence, and inserting the entire object in whichever index the insertion tree specifies. These nodes would not commute with other parts of the diagram and would thus add little complexity to the task of counting T_n, provided we could count M_n^i.

On a different note, we observe that winding sequences partition M_n into subsets of non-uniform sizes. We can then define a random variable X_n on these subsets where $P(X_n = \pi) = \frac{\text{\# meanders with winding sequence } \pi}{M_n}$. We can then ask, what is the entropy of X_n? The exact answer to this question of course awaits an enumeration of M_n, but even partial results using extant approximations may be of interest.

Acknowledgments

We would like to thank our advisor Deanna Haunsperger, for her excellent guidance throughout the past two terms; and Eric Egge and Steve Kennedy for graciously giving their time to listen to our ideas. Furthermore, we extend our thanks to Michael and Annette Morrill, and the Carleton Mathematics Department, who helped fund our trip to the 2016 Joint Mathematical Meetings in Seattle. At the JMM, we met with Thomas Hull and we've greatly benefitted from our communications with him.

References

[1] The on-line encyclopedia of integer sequences. Published electronically at `http://oeis.org`.

[2] The power of origami. *Plus Magazine*. `https://plus.maths.org/content/power-origami`.

[3] Patrizio Angelini, Giordano Da Lozzo, and Daniel Neuwirth. Advancements on sefe and partitioned book embedding problems. *Theoretical Computer Science*, 575:71–89, 2014.

[4] Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Martin L. Demaine, Joseph S.B. Mitchell, Saurabh Sethia, and Steven S. Skiena. When can you fold a map? *Computational Geometry*, 29(1):23 – 46, 2004. Special Issue on the 10th Fall Workshop on Computational Geometry, {SUNY} at Stony Brook.

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

[6] Erik D. Demaine, Eric Liu, and Tom Morgan. A polynomial-time algorithm for $2 \times n$ map folding.

[7] Shepherd Engle. Origami, algebra, and the cubic. 2012. `http://buzzard.ups.edu/courses/2012spring/projects/engle-origami-ups-434-2012.pdf`.

[8] P. Di Francesco, O. Golinelli, and E. Guitter. Meanders and the temperley-lieb algebra. *Communications in Mathematical Physics*, 186(1):1–59.

[9] Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing*, 31:601–625.

[10] Seok-Hee Hong and Hiroshi Nagamochi. Two-page book embedding and clustered graph planarity. Technical report, Department of Applied Mathematics & Physics, Kyoto University, 2009.

[11] Thomas C. Hull. The combinatorics of flat folds: a survey. In *Origami^3*. AK Peters, 2002. `http://arxiv.org/abs/1307.1065`.

[12] Thomas C. Hull. Constructing π via origami. 2007. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.136.2586&rep=rep1&type=pdf`.

[13] Thomas C. Hull. Solving cubics with creases: The work of beloch and lill. *The American Mathematical Monthly*, 118, 2011.

[14] Toshikazu Kawasaki. *Roses, Origami & Math*, pages 136–173. Japan Publications Trading Co., 1998.

[15] M. Lacroix. Approaches to the enumerative theory of meanders. `http://www.math.uwaterloo.ca/~malacroi/Latex/Meanders.pdf`.

[16] Robert J. Lang. Angle quintisection. *Robert J. Lang Origami*. `http://www.langorigami.com/article/angle-quintisection`.

[17] Robert J. Lang. Huzita-justin axioms. *Robert J. Lang Origami*. `http://www.langorigami.com/article/huzita-justin-axioms`.

[18] Charles Livingston. *Knot Theory*. The Mathematical Association of America, 1993.

[19] Rahnuma Islam Nishat and Sue Whitesides. Map folding. In *Canadian Conference on Computational Geometry*.

[20] Maurizio Patrignani. Planarity testing and embedding. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 1, pages 1–42. CRC Press, 2013.

[21] Robert J. Lang Patsy Wang-Iverson and Mark Yim. *Origami 5: Fifth International Meeting of Origami Science, Mathematics, and Education*, pages 531–542. CRC Press, 2011.

[22] Robert J. Lang Roger C. Alperin. One-, two-, and multi-fold origami axioms. *Linear Algebra and its Applications*, 2006.

[23] Sarah-Marie Belcastro Thomas C. Hull. Modelling the folding of paper into three dimensions using affine transformations. *Linear Algebra and its Applications*, 348:273–282, 2002.

[24] Mihalis Yannakakis. Embedding planar graphs in four pages. *Journal of Computer and Sytem Sciences*, 38:36–67, 1989.