MATHEMATICAL ORIGAMI OF A SINGLE VERTEX: REGULAR K-POINTS

MATH 203 CAMCOS TEAM - FALL 2014

Abstract. The goal of mathematical origami is to describe various origami structures using precise mathematical language. In this paper we start with the most simple of origami structures, a vertex on a unit disk. Here we describe a different way of counting how many folded states a flat foldable vertex has and how many mountain valley assignments fold it flat. Moreover we explain a new way to describe flat foldable vertices's with Regular K-Points. Lastly we provide a result in flat foldable meshes.

1. Introduction

Origami is commonly thought of as the art of paper folding however the study of it quickly reveals there are many mathematical characterizations of it. Mathematicians have been studying origami more recently and have discovered some interesting results. A number of authors have discovered properties of the single vertex. In particular, the flat foldability of a single vertex. A common goal of mathematical origami is to classify flat foldablity, or, flat foldable objects. In this paper we discuss results obtained about the flat foldablity of a single vertex and three approaches taken to answer the following questions,

Given a flat foldable vertex and its sector angles:

(1) How many mountain valley assignments will fold flat?
(2) How many distinct folded states are there?

In the first two attempts to answer these questions we generated random sets of flat foldable vertices and tested them. In the third approach we developed a new theory called Regular k-points which exhausts all possible forms of a single vertex. However the information gained from testing the random sets is useful and provides motivation for regular k-points.

2. Single Vertex

We define a single vertex to be a vertex centered on the unit disk with a number n (the degree of the vertex) of straight lines extending from the vertex to the edges of the disk. A single vertex which folds flat is called a flat foldable single vertex. We call a flat folded form of the vertex a folded state. Given a folded state of a vertex, each line is creased in one of two ways, a mountain fold or valley fold. Assigning to each line either a mountain fold or

Date: February 11, 2015.

valley fold is called a mountain valley assignment. We present a few basic theorems relating to necessary and sufficient conditions for flat-foldablity of a single vertex.

Theorem 2.1 (Kawasaki)**.** *Let v be a single vertex of degree $2n$ and let $\alpha_1, \alpha_2, \ldots, \alpha_{2n}$ be the sector angles of v. Then v is flat foldable if and only if*

$$\alpha_1 + \alpha_3 + \cdots + \alpha_{2n-1} = \pi, \quad \alpha_2 + \alpha_4 + \cdots + \alpha_{2n} = \pi.$$

Theorem 2.2 (Maekawa)**.** *Let v be a flat foldable single vertex. Then a mountain valley assignment that flat folds v must satisfy*

$$M - V = \pm 2$$

where M is the number of mountain folds and V is the number of valley folds.

We will refer to a flat foldable single vertex as a k-point, where k stands for Kawasaki, and describe it as $v = (\alpha_1, \alpha_2, \ldots, \alpha_{2n})$, listing the sector angles counter-clockwise in order. Notice there must be an even number of angles due to Maekawa's condition.

Given a k-point, a common way is to study is to look a folded state from the side. For example take the k-point $v = (\alpha_1, \alpha_2, \ldots, \alpha_{2n})$ and one of its folded states as in Figure 1.

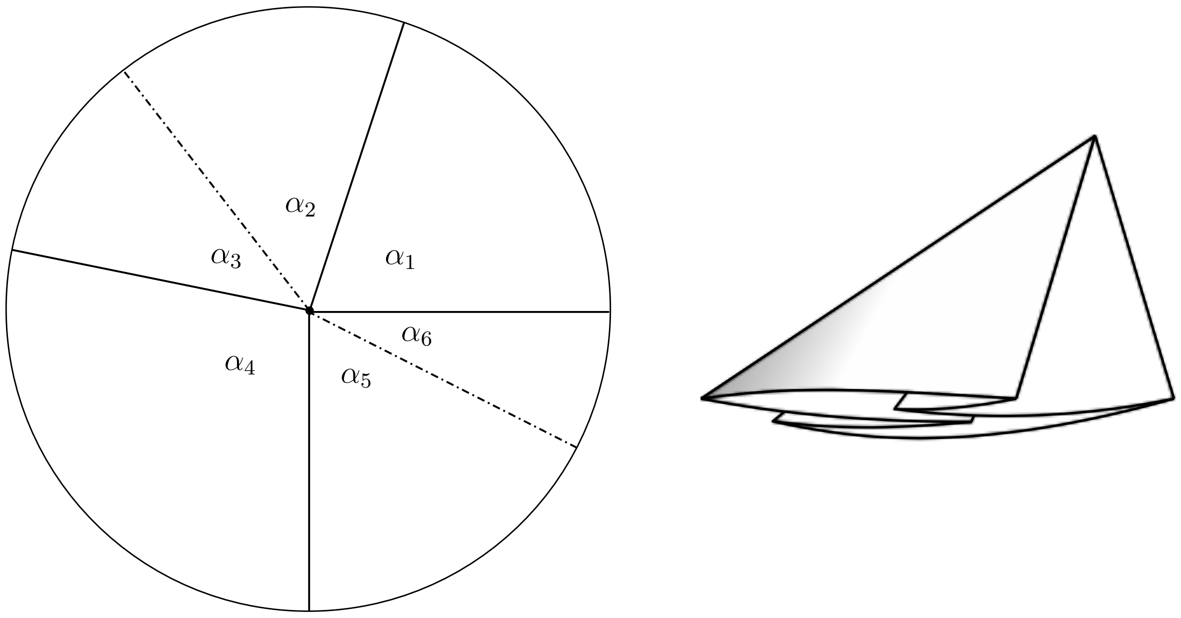


Figure 1.

To get a better idea of what is going on inside, we draw a sector path. To draw the corresponding sector path for this folded state, draw the sector angles as horizontal line segments in the plane evenly spaced apart. Arrange the line segments so they are stacked in the same order as in the folded state in Figure 1. Make sure appropriate end points line

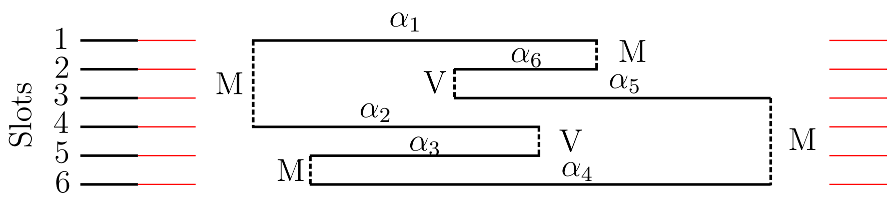


FIGURE 2.

up. Then connect the appropriate line segments with dotted lines. We should be left with something as in Figure 2.

The sector path allows us to tell two different folded states apart with ease. Also the sector path shows the mountain valley assignment. We compactly represent this sector path as a permutation. We can represent Figure 2 simply by the permutation $\rho = (1\ 4\ 5\ 6\ 3\ 2)$ in cycle notation. The permutation ρ represents the order in which we pass through each slot as we traverse along the sector path starting with α_1 moving to α_2. We call ρ the slot permutation. Since the permutation always starts with the slot number that α_1 is in and moves to the slot number that α_2 is in, ρ inherently describes where to put each sector angle.

$$\varphi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 4 & 5 & 6 & 3 & 2 \end{pmatrix}$$

α_1 goes in slot 1, α_2 goes in slot 4, and so on. We represent this with the sector permutation φ. A permutation of the numbers $\{1, 2, \ldots, 2n\}$ such as ρ, represents a possible sector path for v and hence a folded state. Conversely, a folded state of v can be represented by a permutation of the numbers $\{1, 2, \ldots, 2n\}$. Here is another example with the same k-point v and a different permutation. Lets start with a permutation that we know represents a folded state of v and obtain its sector path. The sector path for the permutation $\rho' = (3\ 4\ 5\ 6\ 1\ 2)$ is shown in Figure 3

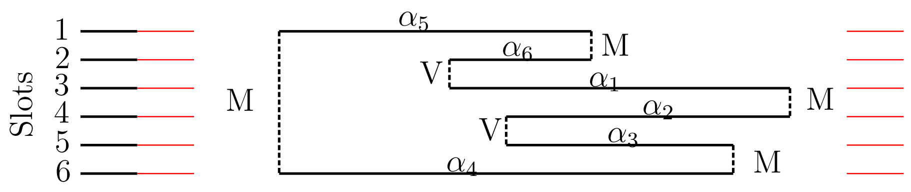


FIGURE 3.

The permutation $\rho' = (3\ 4\ 5\ 6\ 1\ 2)$ implies the sector permutation φ'

$$\varphi' = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 1 & 2 \end{pmatrix}.$$

Which tells us where to put the sector angles. However not every permutation corresponds to a folded state of v. In fact a majority of permutations result in an impossible sector path. Consider the sector path obtained by the permutation (1 3 4 6 5 2) shown in Figure 4.

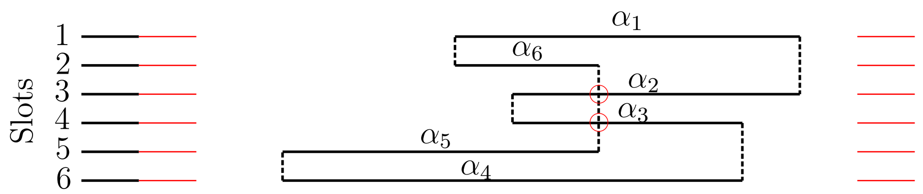


FIGURE 4.

The points where the path intersects itself means the the paper would have to pass through itself to flat fold. Hence this is not a folded state of v.

Going back to the permutation ρ' and its corresponding folded state of v, we can obtain the mountain valley assignment from φ' by calculating φ'^{-1}.

$$\varphi'^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 6 & 1 & 2 & 3 & 4 \end{pmatrix}.$$

With the permutation φ'^{-1} we can see how the sector angels are physically arranged. That is the order of the angles from top to bottom is $\alpha_5, \alpha_6, \alpha_1, \alpha_2, \alpha_3, \alpha_4$. We represent this with the permutation $\psi = (5\ 6\ 1\ 2\ 3\ 4)$ in cycle notation and define it to be the sector arrangement. The sector arrangement is obtained by taking the bottom row of numbers as a cycle in the inverse sector permutation. We know that α_1 connects to α_2 and α_6, with this information and ψ we can deduce the mountain valley assignment in the process described below.

Suppose we have a sector arrangement $\psi = (x_1\ x_2\ ...x_{2n})$ for some n. Now for each x_j, consider the set $\{x_i : i > j\}$ for $i, j \leq 2n$. Suppose α_{x_j} is between α_{x_m} and α_{x_k}. Then if $x_m \in \{x_i : i > j\}$, we can determine what type of crease we have between α_{x_j} and α_{x_m}. If x_j is the same parity as x_1, then the crease will be a mountain. Otherwise, the crease will be a valley. Similarly if $x_k \in \{x_i : i > j\}$, we can determine the type of crease between α_{x_j} and α_{x_k}. Once this is done for all j, we will have the correct mountain valley assignment for the given permutation.

Let's return to the example arrangement $\psi = (5\ 6\ 1\ 2\ 3\ 4)$. Using the process described above, we find that the corresponding mountain valley assignment is $\alpha_1 M \alpha_2 V \alpha_3 M \alpha_4 M \alpha_5 M \alpha_6 V$, or $MVMMMV$. Since there are four mountains and two valleys, this permutation's mountain valley assignment satisfies Maekawa, but that isn't always the case. It turns out that some permutations result in mountain valley assignments that don't satisfy Maekawa.

Suppose we have the slot permutation ρ with sector arrangement $\psi = (x_1\ x_2\ ...\ x_{2n})$. If x_1 and x_{2n} have the same parity, then there will be crossing in the sector path. For example, consider the sector arrangement $\psi = (1\ 2\ 5\ 4\ 6\ 3)$. As we see in Figure 5, there is an unavoidable crossing in a sector path with this permutation.

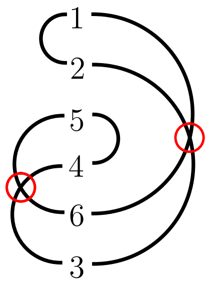


FIGURE 5.

This implies that the paper would have to pass through itself in order to flat fold. Suppose now that $n > 3$. If either $x_1, x_2, ..., x_{n-1}$ or $x_{n+1}, x_{n+2}, ..., x_{2n}$ all have the same parity, then the resulting MV-assignment will not satisfy Maekawa. For example, consider the permutation with the sector arrangement (1 5 3 2 6 7 4 8). Using the method described above, we see that the mountain valley assignment for this permutation will be $MMMMMVMM$, resulting in $M - V = 7$.

Hence we can eliminate some permutations based off the properties mentioned above when looking for folded states, but of the $(2n)!$ permutations, there are still a lot to check. However we were able to reduce the number of permutations greatly. It turns out the only permutations we need to test are meandric permutations (and variations of them).

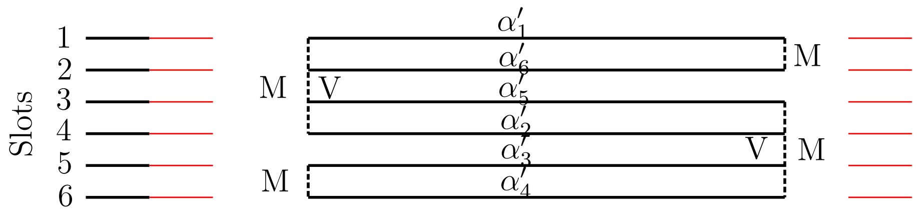


FIGURE 6.

To understand why, consider the k-point v and the sector path in Figure 2. Extend and compress the sector angles so that they are all equal to get a sector path for a new k-point shown in Figure 6. The permutation for the original sector path is the same for this altered version. If we do this for all sector paths of v we obtain a subset of sector paths for the case where all the angles are equal. Hence, given any degree $2n$ k-point v, the set of permutations that flat fold v is a subset of the permutations that flat fold a k-point of degree $2n$ with all equal sector angles.

To determine the set of permutations that correspond to a flat folding of a k-point with all equal angles we introduce meanders and meandric permutations.

Definition 2.3. *A meander is a self avoiding closed curve that intersects a line a number of times.*

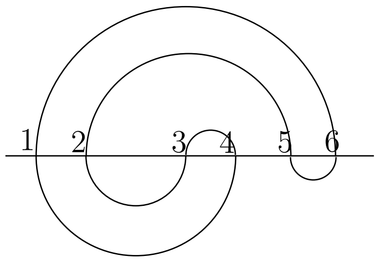


Figure 7. A Meander

If we label the intersection points of a meander 1 to $2n$ starting on the left with 1 and going in order, we obtain a meandric permutation by starting with 1 and moving up along the curve and writing out the intersections we pass through until we return to 1. The meandric permutation from the meander in Figure 7 is (1 6 5 2 3 4).

Now take the sector path from Figure 6 and re-draw the dotted lines as half circles then draw a line down the middle and label the intersections.

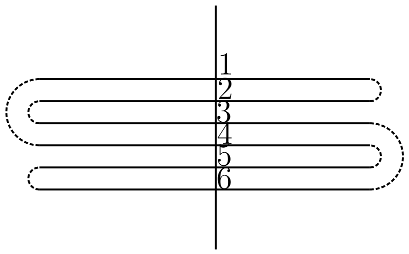


Figure 8. A Meander

We are left with a meander and the corresponding meandric permutation (1 4 5 6 3 2). This permutation can be thought of as the order in which we pass through each slot as we traverse along the meander. Hence, any sector path for a k-point with all equal angles can be seen as a meander. It follows that the only permutations that will yield a valid sector path for a k-point with all equal angles are meandric permutations. Hence, we need only use meandric permutations when looking for a valid sector path for a k-point in general. However we use variations of the meandric permutations as well. A meandric permutation requires that α_1 be in slot 1 (since a meadric permutation starts with 1). Given the meandric permutation $\rho = (1\ 4\ 5\ 6\ 3\ 2)$, we have two other corresponding permutations to consider, $(2\ 1\ 4\ 5\ 6\ 3), (3\ 2\ 1\ 4\ 5\ 6)$. These are obtained by placing 1 in the first three positions. In general for a meandric permutation of length $2n$, we want the permutations obtained by placing 1 in the first n positions. Though these are considered to be the same as permutations, they have different corresponding sector permutations. Which results in more possible sector paths and thus folded states. Hence, for a k-point of degree $2n$ we test a total of $n \cdot M_n$ permutations where M_n is the number of meandric permutations of length $2n$.

We wrote a program called "Fold" that takes a k-point and outputs the number of folded states by testing said permutations. The program also determines the mountain valley assignment for each folded state and outputs the number of distinct mountain valley assignments. The details of this program can be found in the programs section.

With this program our goal was to test a large number of k-points with the same degree and determine the distribution for the number of folded states and mountain valley assignments that occurred in hopes of finding a pattern.

3. Approach 1

The first approach to our problem requires generating a large set of random k-points. To do this we wrote a program called, "The Random k-point Generator" which is described in the programs section however we mention some important aspects of it here. To generate a large amount of random k-points, it is useful to use integers rather than floating point numbers. In a k-point the alternate sum must be π however when generating random k-points we use integers and refer to the alternate sum as σ. To obtain the sector angles one can convert to degrees or radians. When generating a large amount of random k-points we first chose a large σ. For example, here is a random k-point with $\sigma = 500$,

$$v = (48, 6, 135, 203, 317, 291).$$

In degrees v is $(17.28, 2.16, 48.6, 73.08, 114.12, 104.76)$. The larger σ is, the larger the population size is for picking random k-points. Note that for a given σ this population size can be calculated which we omit. For the first test we generated 10,000 random k-points of degree 6 with $\sigma = 500$ then checked to see how many folded states and mountain valley assignments each had. Below are the results

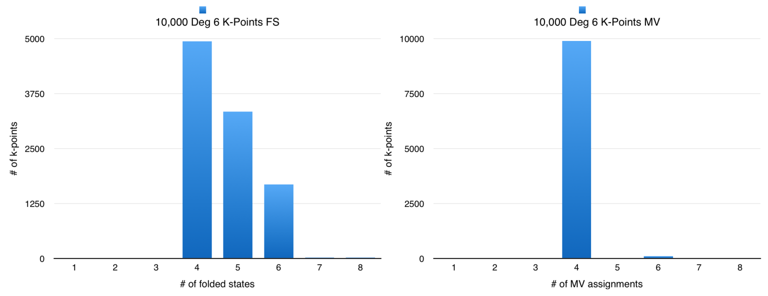


Figure 9.

The first graph is a histogram of the folded states and the second graph a histogram of the mountain valley assignments. In the first graph we see that almost half of the k-points had 4 folded states. In the second graph almost all of the k-points had 4 mountain valley assignments. It seem that we are generating the same ,"type" of k-point over and over again though it should be random.

For the second test we generated 10,000 random k-points of degree 8 with $\sigma = 500$ then checked to see how many folded states and mountain valley assignments each had. Below are the results

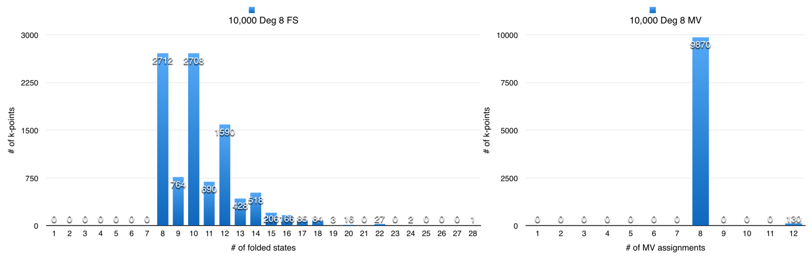


Figure 10.

The first graph is a histogram of the folded states and the second graph a histogram of the mountain valley assignments. We are beginning to see more variety in the folded states however almost all of the k-points have 8 mountain valley assignments. This lead us to believe that using a large σ to generate k-points may be contributing to this. With a large σ there is a high probability of generating k-points with all distinct angles while k-points with some equal angles are not represented at all. In other words, by using a smaller σ there is a higher probability of generating k-points with two or more equal angles. The next section discusses a method of choosing a σ that depends on the degree of the k-point.

4. Approach 2

We now turn our attention to testing random k-points with a small σ. When choosing σ, we want to pick the largest σ that is necessary. In order to do this, we need to consider what we call columns. Consider the k-point $v = (500, 440, 19, 169, 165, 75)$ with $\sigma = 684$ and a folded state represented by the sector path on the left of Figure 14. If we place this sector path in the plane we can mark the x values at which each crease line occurs. We refer to these as columns represented by the red lines in Figure 14 (starting α_1 at $x = 0$). It turns out that if we scale these columns to be unit distance apart from each other, the k-point we are left with works under the exact same permutations. This new k-point $\overline{v}$ on the right of Figure 14 is $\overline{v} = (3, 2, 4, 3, 2, 4)$. The σ for this new k-point is $\sigma = 9$.

In fact, we can calculate the largest σ needed to capture all the folded states for any k-point of degree $2n$. For a k-point of degree $2n$ we can have at most $2n$ columns. This occurs when each crease in the sector path falls on a unique x value in the plane. Suppose we have columns at $0, 1, 2, ..., 2n-1$. Let $\alpha_1 = 2n-1$. This will be our largest segment since it spans all of the columns. Now for α_2 to land on a different column, we need it to be less than 2n-1. Since we are trying to find the largest σ that we need let $\alpha_2 = 2n-2$. Continuing this process we end up with $\alpha_i = 2n-i$ for $1 \leq i \leq 2n-1$. In order for α_{2n} to

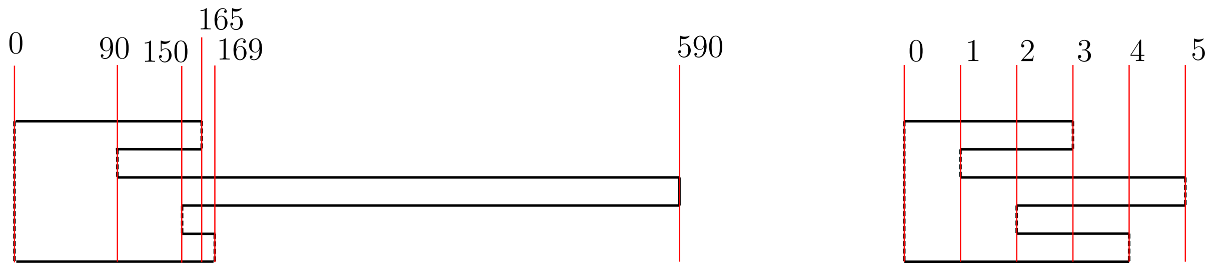


FIGURE 11.

connect with α_1, we must have $\alpha_{2n} = n$. Therefore,

$$2\sigma = (2n-1) + (2n-2) + ... + 2 + 1 + n = \frac{(2n-1)(2n)}{2} + n = 2n^2 \quad (4.1)$$

$$\sigma = n^2 \quad (4.2)$$

With this new σ we generated random k-points and tested them. This resulted in a larger variety of folded states and mountain valley assignments compared to the previous approach. Figure 12 confirms this. Here we tested 5,500 k-points of degree 8 with $\sigma = 16$. However the idea of looking at columns illuminated a new theory of k-points which we call Regular k-points that allowed us to classify all types of k-points for small n.

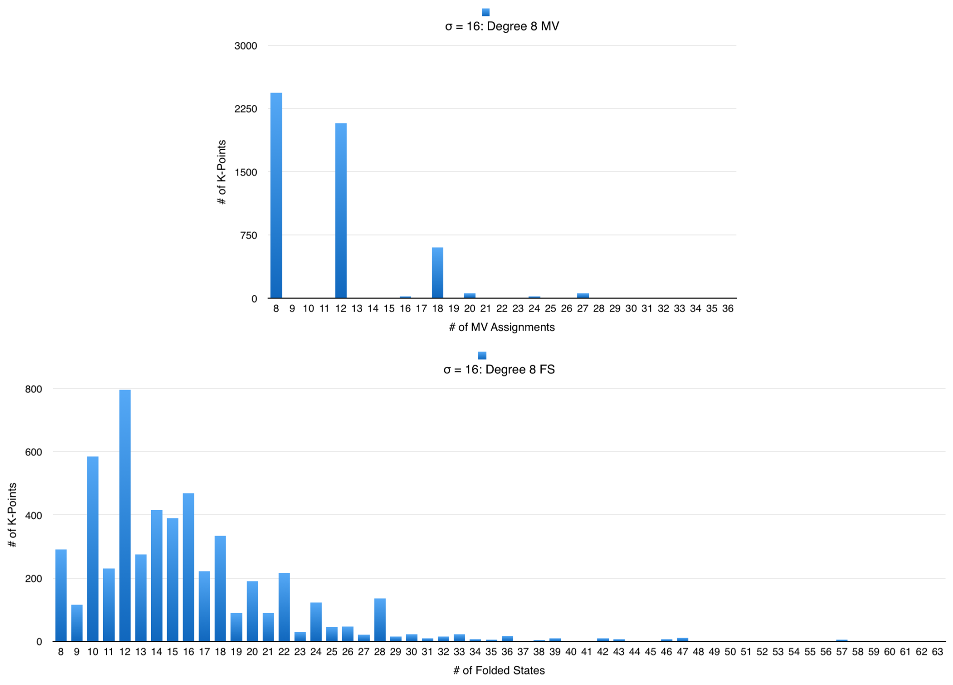


FIGURE 12.

5. Approach 3: Regular k-points

In this section we will introduce a new characterization of a flat foldable vertex called a column sequence which ultimately provides a means to efficiently determine if two k-points flat fold under the same permutations. To begin, we will revisit the definition of a k-point. Note that since the first and last sectors of a vertex are adjacent, index arithmetic is always evaluated modulo 2n so that 1 comes after 2n and 2n comes before 1.

Definition 5.1. *A finite sequence of positive real numbers $\boldsymbol{x} = (x_1, x_2, ..., x_{2n})$, $n \in \mathbb{N}$, is a k-point of degree 2n if the sum of even-indexed terms equals the sum of odd-indexed terms. We call this alternate sum σ, or $\sigma(\boldsymbol{x})$ when a k-point $\boldsymbol{x}$ is given.*

In practice we use k-points with integer values and convert to degrees or radians need be. However we provide a motivating example with rational numbers. Consider the k-point $\mathbf{x} = (50, 44, 1.9, 16.9, 16.5, 7.5)$ with $\sigma(\mathbf{x}) = 50 + 1.9 + 16.5 = 68.4$.

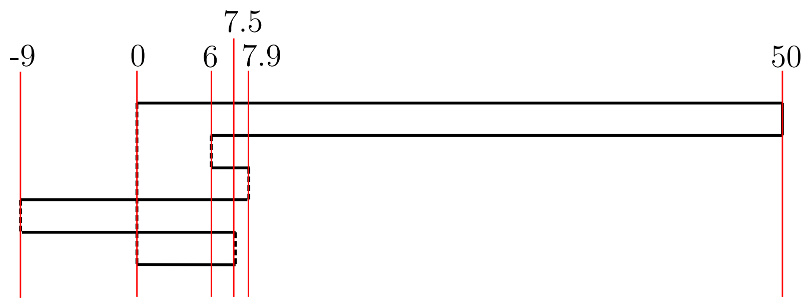


Figure 13.

Figure 13 shows a diagram for $\mathbf{x}$ using an arbitrary permutation. For this discussion, the specific permutation used is not of interest so the slot numbers are omitted. Instead, we want to look at the vertical lines, or "columns", as well as the numbers above them, the "column coordinates." The horizontal segments represent sectors of a folded vertex having angle measures proportional to the terms of $\mathbf{x}$. Columns mark the segments' endpoints which correspond to creases in the folded vertex. Column coordinates are the result of placing these segments along a number line so that the lengths of the segments match the terms of $\mathbf{x}$.

To determine the column coordinates, we always begin by choosing the left endpoint of x_1 to be 0. Its right endpoint is chosen to be 50 so that the difference between those coordinates is the desired length, $50 - 0 = 50 = x_1$. Next, the right endpoint of x_2 should be aligned with x_1 at 50 and we choose the column coordinate of its left endpoint to be 6. Thus the length of the segment is 50-6=44, which is the value of x_2 as desired. Similarly, x_3 will go from 6 up to 7.9 to achieve a length of 1.9, x_4 will go from 7.9 down to -9 for a length of 16.9, x_5 will go from -9 up to 7.5 for a length of 16.5, and x_6 will go from 7.5 down to 0 for a length of 7.5. The last endpoint will always make it back to 0 since the Kawasaki condition, as it applies to the diagram, requires the total rightward displacement to equal the total leftward displacement.

These numbers are uniquely determined by the k-point, independently from the permutation used. However, a vertex is not characterized by a set of column coordinates alone, we also need information about the order in which the columns are reached by the sectors. This leads us to

Definition 5.2. *A finite sequence of real numbers $(c_1, ..., c_{2n})$, $n \in \mathbb{N}$, is a column sequence of degree 2n if $c_1 = 0$ and every even-indexed term is greater than both of its neighboring odd-indexed terms, with the first and last terms considered to be neighbors as well.*

In order to relate this definition to our study, we move on to define the one-to-one correspondence between k-points and column sequences. The column sequence corresponding to a given k-point $\mathbf{x} = (x_1, x_2, ..., x_{2n})$ is determined by the function

$$C(\mathbf{x}) = (0, x_1, x_1 - x_2, x_1 - x_2 + x_3, ..., \sum_{j=1}^{i} (-1)^{j+1} x_j) \text{ for } 1 \leq i \leq 2n-1.$$

Note that $C(\mathbf{x})$ satisfies the definition of a column sequence by construction since the first term is 0, every even-indexed term c_i is the same as c_{i-1} plus some positive number from the k-point, and c_{i+1} is the same as c_i minus some positive number. The k-point corresponding to a given column sequence $\mathbf{c} = (c_1, c_2, ..., c_{2n})$ is determined by the function

$$K(\mathbf{c}) = (c_2 - c_1, c_2 - c_3, c_4 - c_3, c_4 - c_5, ..., c_{2n} - c_{2n-1}, c_{2n} - c_1).$$

Each term is positive, since the even-indexed c_i's are greater than both c_{i-1} and c_{i+1}. So verifying that $K(\mathbf{c})$ is a k-point comes down to the alternate sums being equal:

$$(c_2 - c_1) + (c_4 - c_3) + ... + (c_{2n} - c_{2n-1}) = (c_2 - c_3) + (c_4 - c_5) + ... + (c_{2n} - c_1),$$

which is a simple matter of rearranging terms. We will use the example k-point from before to demonstrate these definitions. Given $\mathbf{x} = (50, 44, 1.9, 16.9, 16.5, 7.5)$, we have

$$\begin{aligned} C(\mathbf{x}) &= (0, 50, 50 - 44, 50 - 44 + 1.9, 50 - 44 + 1.9 - 16.9, 50 - 44 + 1.9 - 16.9 + 16.5) \\ &= (0, 50, 6, 7.9, -9, 7.5). \end{aligned}$$

The terms of $C(\mathbf{x})$ are the column coordinates in the diagram for $\mathbf{x}$ (see figure 1) in the exact order that they are traversed by the sectors. Now, we will use the $K(\mathbf{c})$ function to get back a k-point:

$$\begin{aligned} K(C(\mathbf{x})) &= (50 - 0, 50 - 6, 7.9 - 6, 7.9 - (-9), 7.5 - (-9), 7.5 - 0) \\ &= (50, 44, 1.9, 16.9, 16.5, 7.5). \end{aligned}$$

Conveniently, this is the same k-point we started with. In fact, the functions K and C are each others inverse, so this will always be the case. Hence there is a one-to-one correspondence between k-points and column sequences, and we now have two ways to characterize a flat-foldable vertex numerically. The advantage to using column sequences is that we can state a well-defined condition that determines if a k-point and permutation yield a folded-state.

Connectibility: Let $\mathbf{c} = (c_1, c_2, ..., c_{2n})$ be a column sequence and let $\varphi : I \to I$ be a permutation of the index set $I = \{1, ..., 2n\}$. For convenience, let $a(i) = min(\varphi(i), \varphi(i+1))$ and $b(i) = max(\varphi(i), \varphi(i+1))$ for any $i \in I$. Let j' denote $\varphi^{-1}(j)$ and j'' denote $\varphi^{-1}(j) + 1$. We say that the stack $\langle \mathbf{c}, \varphi \rangle$ is unconnectible if there exists $i \in I$ and $j \in I$, with $a(i) < j < b(i)$, that satisfy either of two conditions:

(i) ***Crossing at i-j:*** $min\,(c_{j'}, c_{j''}) < c_{i+1} < max\,(c_{j'}, c_{j''})$

(ii) ***Interleaving at i-j:*** $c_{i+1} = c_{j''}$ and either $\varphi(j'') < a(i)$ or $\varphi(j'') > b(i)$.

The stack $\langle \mathbf{c}, \varphi \rangle$ is connectible if it is not unconnectible. Crossing at i-j implies that the crease connecting sectors i and $i+1$ intersects the sector in slot j. Interleaving is essentially the same conflict with the crease connecting sectors i and $i+1$ overlapping the crease connecting sectors j' and j''. Two k-points $\mathbf{x}$ and $\mathbf{y}$ are connectimorphic, denoted $\mathbf{x} \sim \mathbf{y}$, if they are connectible under the same permutations. Clearly $\sim$ is an equivalence relation.

Example: Consider the k-point $\mathbf{x} = (50, 44, 1.9, 16.9, 16.5, 7.5)$ and another k-point $\mathbf{y} = (4, 3, 2, 4, 3, 2)$, both with the permutation $\rho = (1\ 2\ 3\ 4\ 5\ 6)$.

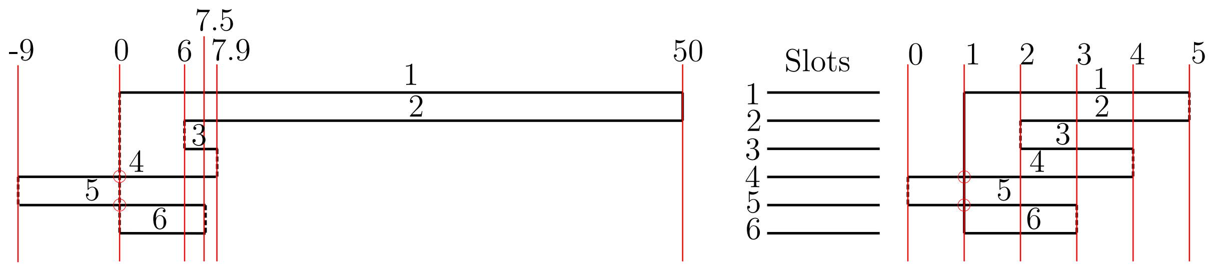


FIGURE 14.

We see that both stacks have crossing at 6-4 and 6-5, meaning the crease following segment 6 intersects the segments in slot 4 and slot 5. For $\langle \mathbf{x}, \rho \rangle$ and $\langle \mathbf{y}, \rho \rangle$, the crossing at 6-4 is due to the inequalities $-9 < 0 < 7.9$ and $0 < 1 < 4$, respectively. Both stacks have crossing in the same place simply because sector 4 extends from the first column on the left to the second-to-last column, with sectors 1 and 6 attempting to connect along a column between those two. The purpose of this example is to observe that the lengths of the sectors and distances bewteen the columns are more information than what is needed to determine if a k-point is connectible under a specific permutation.

The diagram for $\langle \mathbf{x}, \rho \rangle$ has a column far out to the right and several columns crammed together in the middle, which accurately portrays the k-point but does not help when checking connectibility. Notice how we could relabel the column coordinates for $\langle \mathbf{x}, \rho \rangle$ with the numbers $0, 1, 2, 3, 4$, and 5, then make the spacing between the columns unit distance and the diagram would look the same as $\langle \mathbf{y}, \rho \rangle$. This change of column coordinates preserves all crossings, interleavings, and lack thereof. For this reason, under any other permutation, $\mathbf{x}$ and $\mathbf{y}$ will both be connectible or both unconnectible, so $\mathbf{x} \sim \mathbf{y}$.

The k-point $\mathbf{y}$ also has an important property that all of its columns have unit spacing between them and its column coordinates are consecutive integers beginning with 0. Because of this, we call $\mathbf{y}$ a regular k-point, and $C(\mathbf{y})$ a regular column sequence, which we give as

Definition 5.3. *Let $\mathbf{c}$ be a column sequence and m be the maximum of the terms in $\mathbf{c}$. We say that $\mathbf{c}$ is a regular column sequence and write $\bar{\mathbf{c}}$ if m is a positive integer and the set of distinct terms in $\mathbf{c}$ is exactly the set of integers $\{0, 1, 2, ..., m\}$. A k-point is a regular regular k-point if its column sequence is regular.*

Regular k-points are, in a sense, the nicest k-points because their sector angles and column coordinates are generally small integers that are easy to work with by hand or computer and their diagrams naturally live on a grid. Furthermore, it is not hard to show that every k-point is connectimorphic to some regular k-point.

Regularization: Simply put, the regularization of a k-point is a change of column coordinates where the least coordinate is replaced by 0, the next lowest is replaced by 1, and so on up to the greatest, which is replaced by $k - 1$ when there are k many distinct column coordinates. This can be expressed more precisely as follows.

Let $\mathbf{c} = (c_1, c_2, ..., c_{2n})$ be a column sequence of degree 2n and let $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_k\}$ be the set of distinct column coordinates (i.e. terms in $\mathbf{c}$) with the indices chosen so that $\gamma_1 < \gamma_2 < ... < \gamma_k$. Define a change of coordinates $f : \Gamma \to \mathbb{Z}$ by $f(\gamma_i) = i - 1$ for all i from 1 to k. The *regularization* of $\mathbf{c}$, denoted $\bar{\mathbf{c}}$, is the column sequence $\bar{\mathbf{c}} = (f(c_1), f(c_2), ..., f(c_{2n}))$.

Since the regularization of a column sequence preserves pairwise inequalities between its terms, connectibility under each permutation is preserved as well, meaning $\mathbf{c}$ and $\bar{\mathbf{c}}$ are connectimorphic. That means we can just check connectibility under each permutation for $\bar{\mathbf{c}}$, then conclude that the results apply to $\mathbf{c}$ as well. It is not hard to see that regularization is defined for any column sequence, so every column sequence is connectimorphic to some regular column sequence.

However, the process of regularization is not one-to-one, meaning the connectibility results for $\bar{\mathbf{c}}$ actually apply to many column sequences. These properties suggest a strategy to determine if two given column sequences $\mathbf{a}$ and $\mathbf{b}$ are connectimorphic. Using transitivity of $\sim$, we find that $\bar{\mathbf{a}} = \bar{\mathbf{b}}$ implies $\mathbf{a} \sim \mathbf{b}$. That is, if two column sequences have the same regularization, then they represent two k-points that flat fold under the same permutations.

How many regular column sequences are there?: There clearly are uncountably many column sequences for any degree greater than 2, so checking connectibility under each permutation for every column sequence of a fixed degree is impossible. On the other hand, we have now shown that restricting this task to only checking regular column sequences would still be exhaustive. That is, the results for an arbitrary $\mathbf{c}$ are implicit in the results for $\bar{\mathbf{c}}$. So how many do we really need to check?

It turns out, there are only finitely many regular column sequences of a fixed degree $2n$, a number we will denote $\mathscr{R}_{2n}$. Since a regular column sequence must contain all integers from 0 to some m and there are only $2n$ terms, it follows that $m \leq 2n-1$. The maximal cases when $m = 2n-1$ admit $2n$ choices for each term so, in any case, each of the $2n$ terms has at most $2n$ possibilities. Hence, $\mathscr{R}_{2n}$ is bounded above by $(2n)^{2n}$. While this upper bound grows very fast even for small n, most of the "choices" accounted for are not even column sequences, so $\mathscr{R}_{2n}$ is really much smaller. As we will see, $\mathscr{R}_4 = 3$ while $4^4 = 256$. Finding a closed formula for $\mathscr{R}_{2n}$ as a function of n is still an open problem.

Generating regular column sequences: We wrote a program that outputs the list of all regular column sequences of a given degree through a process of elimination from a larger list. The program consists of nested loops that generate every integer sequence $(c_1, c_2, ..., c_{2n})$ where $c_1 = 0$, $c_1 < c_2 < 2n$, $0 \leq c_3 < c_2$, $c_3 < c_4 < 2n$, and so on. Each sequence is searched for the numbers from 0 to its maximum m and eliminated if any number is missing. A sequence that passes this test is certainly a regular column sequence, however some will still need to be eliminated as duplicates of what we consider to be the same k-points.

Since a vertex does not have a natural first and last angle, there are many ways to pick which one is listed first in its k-point, and then two choices for which way to loop around. These different choices of labeling result in circular shifts and reversals of the terms in the k-point while preserving the adjacency of angles in the original vertex. Of course, these k-point-preserving shifts are passed on to the corresponding column sequence and vice-versa. One more subtle k-point-preserving shift comes from replacing each term c_i in the column sequence with the number $m - c_i$, where m is the maximum term. This has the effect of reflecting the k-point's diagram across a vertical line.

We chose to define a unique representative for each k-point by imposing a lexicographic order on the different shifts of its column sequence and taking the least one. So for each sequence $\bar{\mathbf{c}}$ that passes the regular column sequence test, the program generates all of its shifts and checks that $\bar{\mathbf{c}}$ itself is the least of all, otherwise it is eliminated from the final list.

Questions:

1) It is unknown whether regular column sequence partitions are the same as that of $\sim$ or a slightly finer partitioning. That is, we know that $\bar{\mathbf{a}} = \bar{\mathbf{b}}$ implies $\mathbf{a} \sim \mathbf{b}$, but we only speculate that $\mathbf{a} \sim \mathbf{b}$ implies $\bar{\mathbf{a}} = \bar{\mathbf{b}}$. A counter example would be two distinct regular column sequences that happen to be connectible under all the same permutations. If such

a case exists, then there are multiple regular column sequence partitions within a single equivalence class of $\sim$.

2) What is the closed formula for $\mathscr{R}_{2n}$ as a function of n?

3) Is there an algorithm to directly generate the list of regular column sequences of a fixed degree without using elimination from a larger list?

6. Conclusion/Results

We have shown that to study faltfoldable verticies, it is sufficient to study regular k-points. The fold program takes a k-point and counts the number of folded states and the number of valid mountian valley assignments. With the fold program, we tested all regular k-points for small n. The program then calculates all folded states and all valid mountain valley assignments for each regular k-point for small n. We compiled this data and ran some analyses. In particular, we determined what we call fold famillies.

Recall that: For the degree 4 case, there are 3 regular K-points.

For the degree 6 case, there are 29 regular K-points.

For the degree 8 case, there are 704 regular K-points.

For the degree 10 case, there are ... regular K-points.

Fold Families. Using this information, we formed fold families.

We say that two k-points are members of the same fold family when they have the exact same number of possible folded states and the exact same number of possible mountain valley assignments as one another.

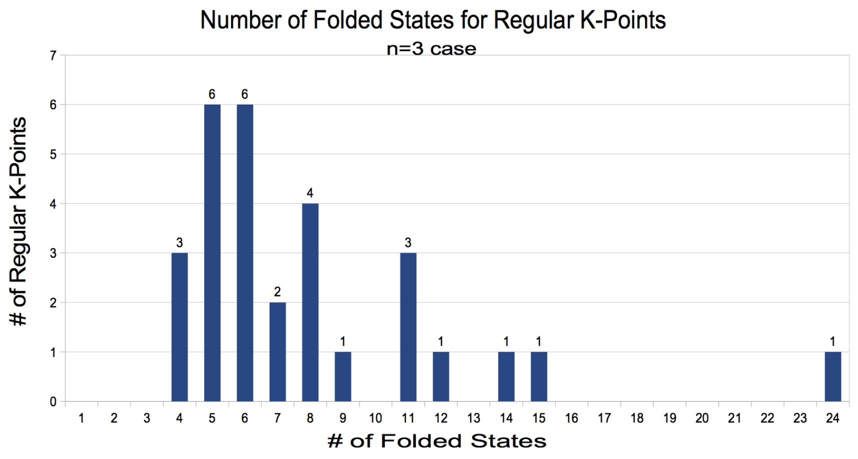
To illustrate this, consider the three distinct k-points, (30, 30, 60, 60, 90, 90), (45, 45, 90, 45, 45, 90), (36, 36, 108, 36, 36, 108). Each of these k-points have distinct regular k-points, (1,1,2,2,3,3), (1,1,2,1,1,2) and (1,1,3,1,1,3) respectively, but they all have exactly 11 possible folded states and there are exactly 9 mountain valley assignments. Thus, these three k-points are all in the same fold family.

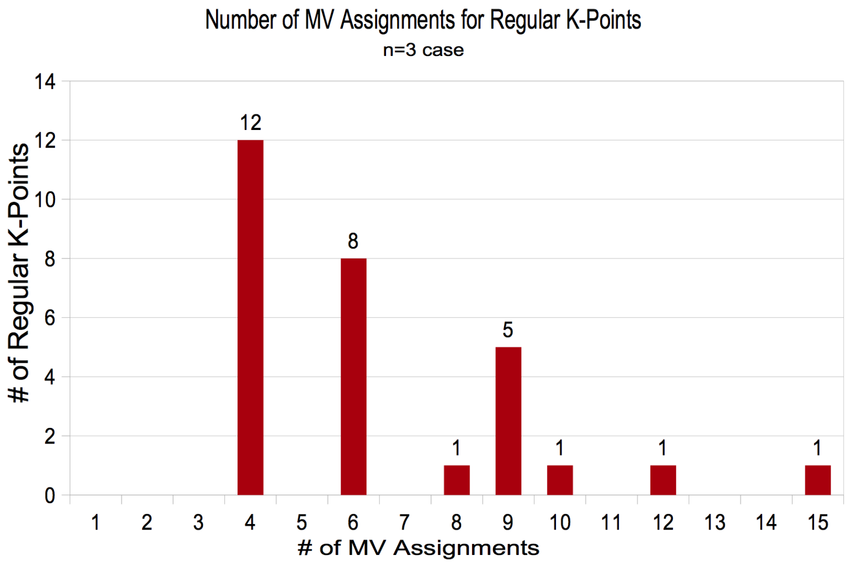
Degree Four Vertices. Of these regular k-points, for degree 4 we found that there are three possibilities for the number of ways to fold such a vertex, that is, there are three possible numbers for folded states, either 4, 3, or 2. Also, there are three possibilities for the number of mountain valley assignments, either 4, 3, or 2.

For degree 4, we found there are three fold families: points that have four folded states and four mountain valley assignments (4,4), points that have three folded states and three mountain valley assignments (3,3), and points that have two folded states and two mountain valley assignments (2,2).

Now, if we pick any arbitrary degree 4 k-point, then it is connectimorphic to one of the three regular k-points of degree 4. As an example, we take a k-point that has the angles (45, 60, 135, 120). This k-point is connectimorphic to the regular k-point (2,1,2,3), which has exactly 2 folded states and 2 mountain valley assignments, hence this k-point has exactly 2 folded states and exactly 2 mountain valley assignments.

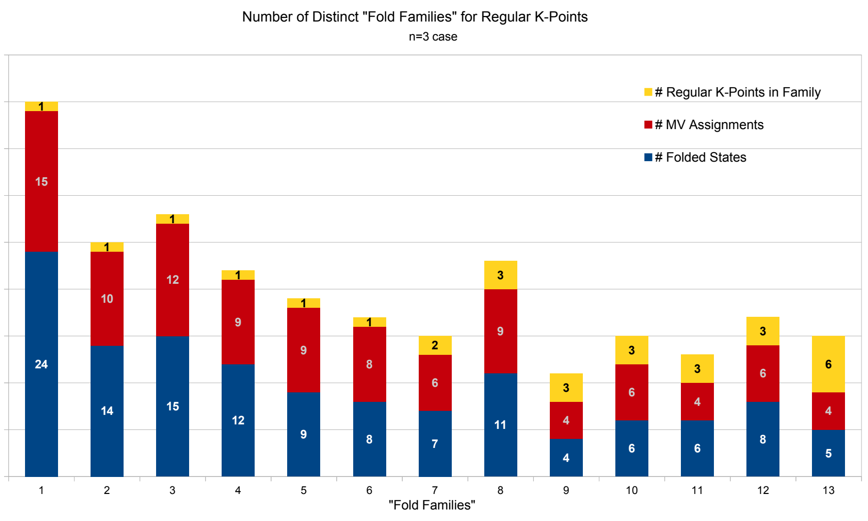
Degree 6 Vertices. Of these regular k-points, for degree 6 we see that there are 11 possibilities for the number of folded states: 4, 5, 6, 7, 8, 9, 11, 12, 14, 15, 24. Also there are 7 possibilities for the number of mountain valley assignments: 4, 6, 8, 9, 10, 12, 15.





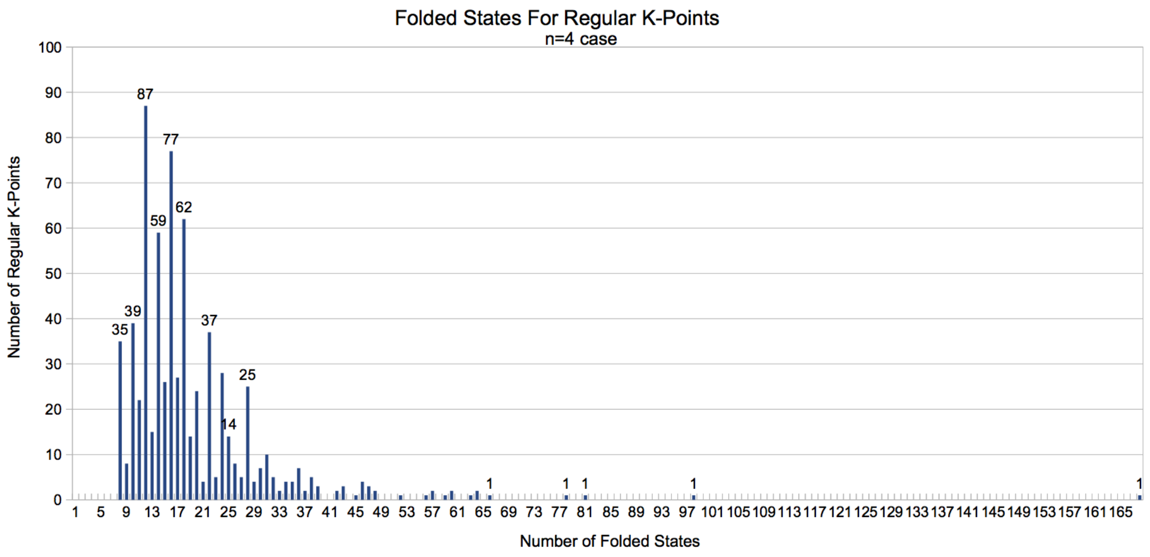
From this data we can see that any arbitrary degree 6 vertex must have at least 4 and no more than 24 folded states. Also, we can say with certainty that there is *no* degree 6 vertex that has exactly 5, 7, 11, 13, or 14 mountain valley assignments.

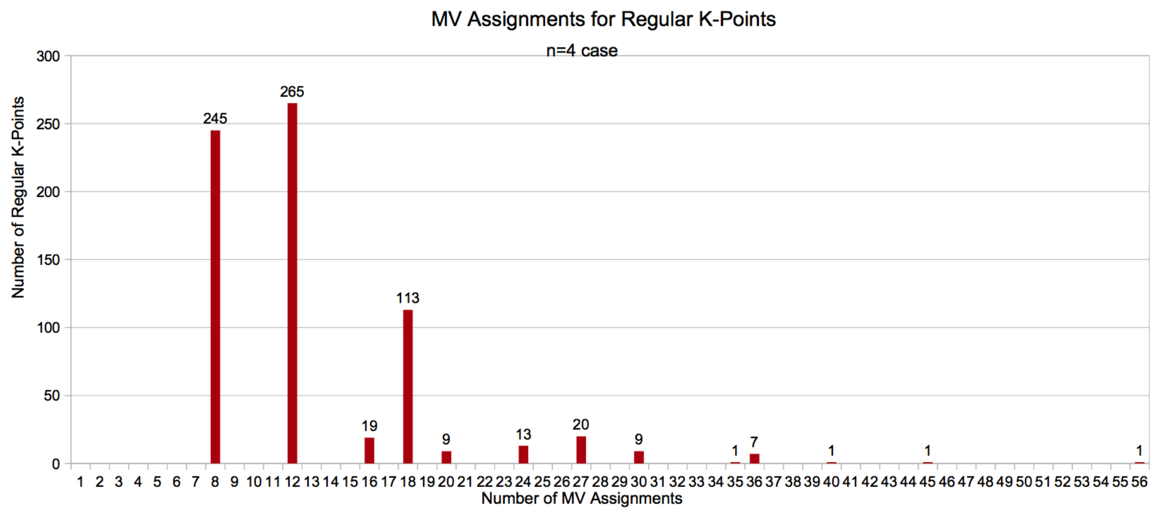
Each of the degree 6 regular k-points falls into one of thirteen fold families, as shown in the following histogram.

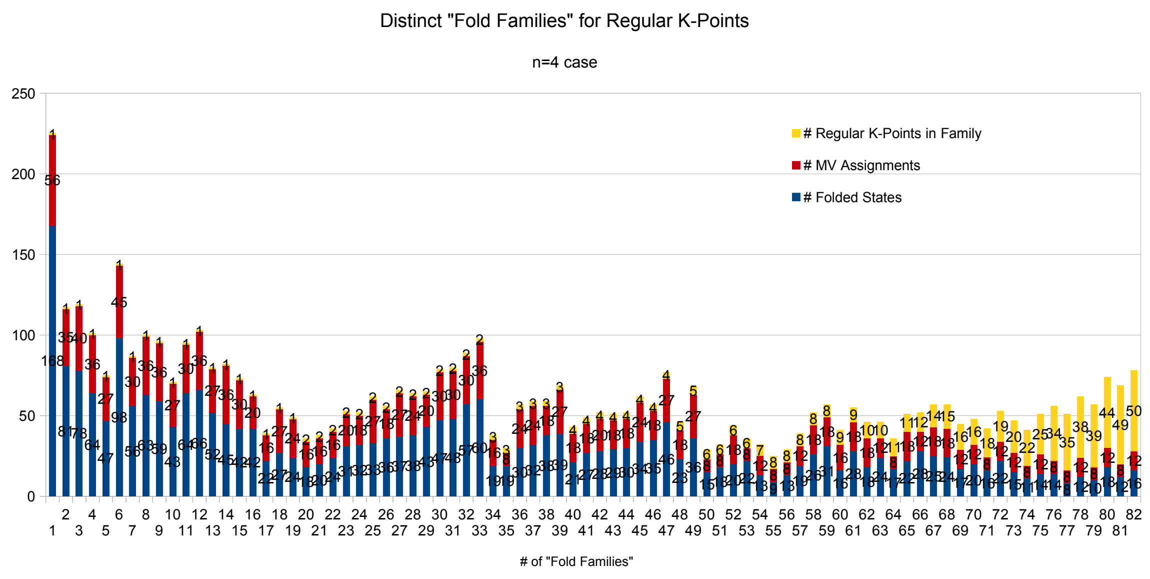


From the fold family analysis, we can say that a k-point with exactly 6 mountain valley assignments has either 6, 7, or 8 folded states, no more, no less, and exactly one of these is true.

Degree 8 Vertices. For degree 8 regular k-points we see that there are 50 possibilities for the number of folded states and there are 13 possibilities for the number of mountain valley assignments (8, 12, 16, 18, 20, 24, 27, 30, 35, 36, 40, 45, 56).

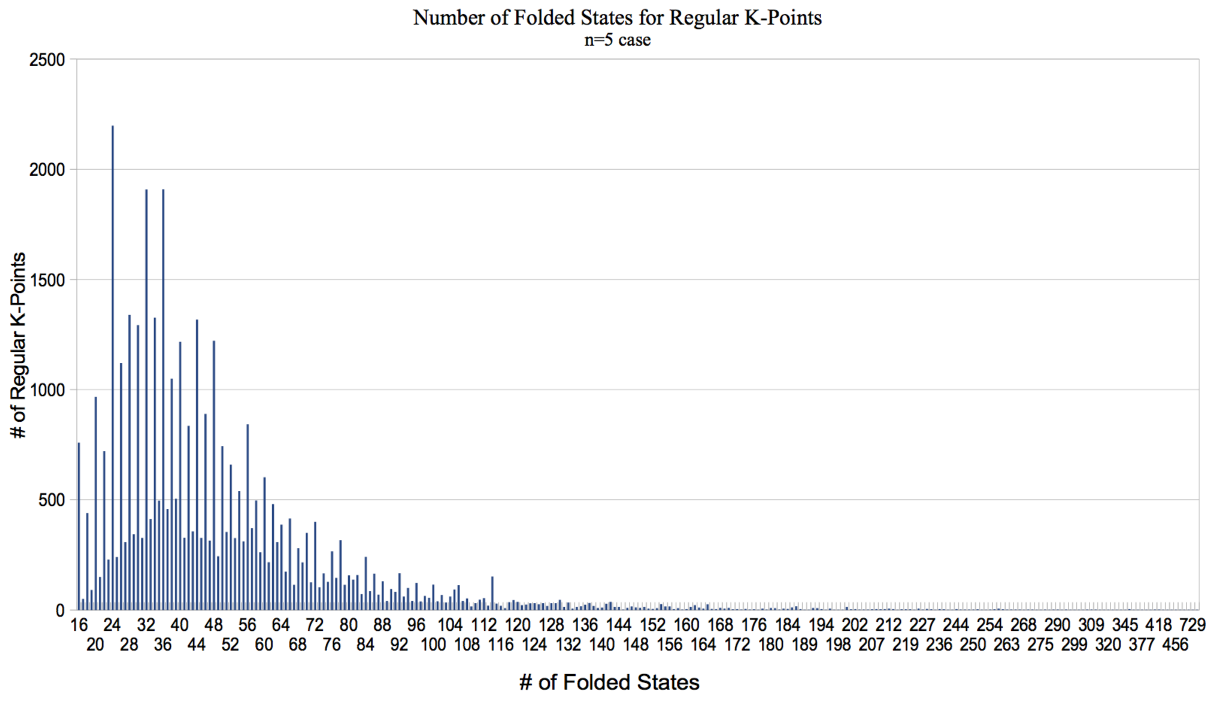



MV Assignments for Regular K-Points
n=4 case
Number of Regular K-Points
Number of MV Assignments
245
265
19
113
9
13
20
9
1
7
1
1
1

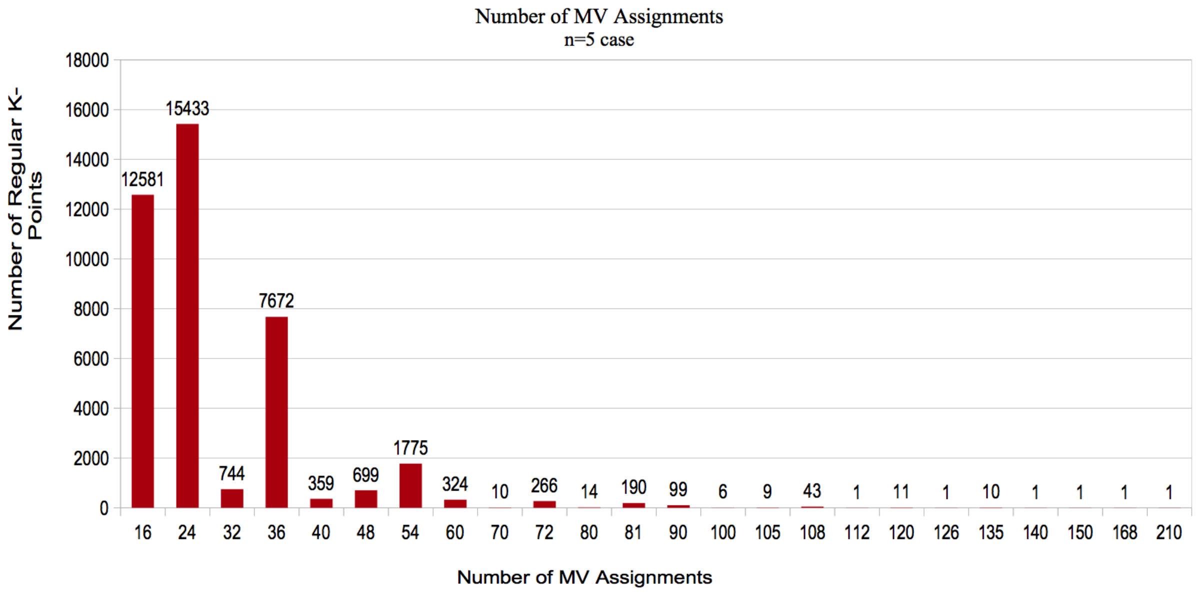

Distinct "Fold Families" for Regular K-Points
n=4 case
# Regular K-Points in Family
# MV Assignments
# Folded States
# of "Fold Families"

Degree 10 k-points. **Folded States**

Degree 10 Vertices



MV Assignments



There are 739 fold families for degree 10 vertices.

Identifying the Maximum Number of Folded States. Recall, from the single vertex section, that the maximum number of folded states for a degree $2n$ vertex is $n \cdot M_n$. Recall also the meandric numbers: $M_1 = 1, M_2 = 2, M_3 = 8, M_4 = 32, M_5 = 262$ From our analysis, we see that this is in fact the case: for degree 4=2· 2 there are at most 4 folded states, for degree 6=2· 3 there are at most 24 folded states, for degree 8=2· 4 there are at most 168 folded states, for degree 10=2· 5 there are at most 262· 5=1310 folded states. The largest number of folded states comes from the case where all sector angels are equal. Here each meandric permutation gives a connectable permutation that leads to a folded state for the vertex.

Future Research. We successfully answered our research question through the use of the Fold program. If we have a given Kawasaki vertex of a relatively small degree (i.e. ≤ 12) we may input the size of the angles into the fold program, which then outputs the number of distinct ways to flat-fold such a vertex as well as the number of distinct mountain valley assignments that flat fold. All this information tells us explicitly how many ways there are to fold our given vertex.

As we analyzed our data, we came up with even more questions. We wondered: Why do certain numbers of folded states/mountain valley assignments show up while others don't? Is there a pattern here that depends on n? How is the number of fold families related to the degree of the vertex?

7. Hull's Theory

In the paper *Counting Mountain-Valley Assignments for Flat Folds* [?], Thomas C. Hull introduces the topics of counting valid mountain valley assignments. Hull uses the fact that for any sequence of angles, the smallest angle will be neighbored by angles that are greater or equal. The equal angles or smallest angle can then be fused, or folded and absorbed by a new angle. The difference of mountains and valleys among these fused angles can be determined by adjusting and using Maekawa's theorem. The result is a mechanism to determine all mountain valley assignments for a single vertex.

Hull's paper includes a theorem which gives a recursive formula for determining the number of valid mountain valley assignments for a given sequence of angles which is stated as

Theorem 7.1. *Let $v = (1, ..., 2n)$ be a flat vertex fold in either a piece of paper or a cone, and suppose we have $\alpha_i = \alpha_{i+1} = \alpha_{i+2} = \cdots = \alpha_{i+k}$ and $\alpha_{i-1} > \alpha_i$ and $\alpha_{i+k+1} > \alpha_i$ for some i and k. Then*

$$C(\alpha_i, ..., \alpha_{2n}) = \binom{k+2}{\frac{k+2}{2}} C(\alpha_1, ..., \alpha_{i-2}, \alpha_{i-1}\alpha_i + \alpha_{i+k+1}, \alpha_{i+k+2}, ..., \alpha_{2n})$$

if k is even, and

$$C(\alpha_i, ..., \alpha_{2n}) = \binom{k+2}{\frac{k+1}{2}} C(\alpha_1, ..., \alpha_{i-1}, \alpha_{i+k+1}, ..., \alpha_{2n})$$

if k is odd.

Here $C(\alpha_i, ..., \alpha_{2n})$ represents the number of valid mountain valley assignments. The paper includes examples of recursively computing the number of mountain valley assignments, but this is not the whole picture. In our research we wanted to calculate all possible distinct numbers of mountain valley assignments. In other words, given k-point of degree $2n$, what are the possible values for the number of valid mountain valley assignments, and how many are there?

In [**?**], this is not explicitly stated, however, a more recent article, *The Flat Vertex Fold Sequences* [**?**], by Hull and Eric Chang, explicitly explains how the numbers of mountain valley assignments are obtained. Thus, the number of numbers of mountain valley assignments are obtained. The following formula found in [**?**] is utilized to produce the values aforementioned.

$SC(1) = 2$ and for $n \geq 2$ we have

$$SC(n) = \left(\bigcup_{k=1}^{n-1} \left(\binom{2n-2k}{n-k} SC(k) \cup \binom{2n-2k+1}{n-k} SC(k)\right)\right) \cup \left\{2\binom{2n}{n-1}\right\}.$$

Here $SC(n)$ is the set of all possible distinct numbers of mountain valley assignments. This recursively generates the list of valid mountain valley assignments for a given degree. The remarkable aspect of this combinatoric model for computing the total list of valid mountain valley assignments for a degree $2n$ vertex is that the numbers agree with the results obtained by our Fold program.

The list of valid numbers of mountain valley assignments, obtained using Hull's formula, for $n = 1, ..., 5$, are:

$$\begin{aligned}
SC(1) =& \{2\} \\
SC(2) =& \{4, 6, 8\} \\
SC(3) =& \{8, 12, 16, 18, 20, 24, 30\} \\
SC(4) =& \{16, 24, 32, 36, 40, 48, 54, 60, 70, 72, 80, 90, 112\} \\
SC(5) =& \{32, 48, 64, 72, 80, 96, 108, 120, 140, 144, 160, 162, 180, 200, \\
& 210, 216, 224, 240, 252, 270, 280, 300, 336, 420\}.
\end{aligned}$$

Thus,

$$|SC(1)| = 1$$
$$|SC(2)| = 3$$
$$|SC(3)| = 7$$
$$|SC(4)| = 13$$
$$|SC(5)| = 24$$

The Fold program counts mountain valley assignments so that $M - V = 2$ where M is the number of mountains and V is the number of valleys. By multiplying each number obtained by the Fold program by 2, we obtain exactly the same results as Hull. With regards to the regular k-points, this confirms our results encompass all possible values. For example, if we run all of the regular k-points of degree $6(n = 3)$ through the Fold program, and create a set M of the numbers of M-V assignments, we get

$$M = \{4, 6, 8, 9, 10, 12, 15\}, \quad \text{and} \quad |M| = 7.$$

So that $M' = \{2m| \; m \in M\} = \{8, 12, 16, 18, 20, 24, 30\} = SC(3)$. The conclusion contains the complete results from the Fold program which may be used to verify that we obtain the same results for computing all the numbers of distinct M-V assignments.

8. Polygon Mesh & The Kawasaki Curve

Suppose we have a mesh, that is, a polygon with a number of interior vertices. In order for the mesh to be flat foldable, we need all of the interior vertices to be Kawasaki. We begin by investigating whether an arbitrary point interior to a polygon is Kawasaki. Next, we identify the locations of all interior Kawasaki points by defining the *Kawasaki space* of a $2n$-gon.

Suppose you have a mesh with $2n$ vertices, $v_1, \ldots, v_{2n}$, such that each v_i is Kawasaki and such that an edge coming off each vertex meets at one point (call it v) interior to the others.
Question: Is v necessarily Kawasaki?
Claim: v is not necessarily Kawasaki.

Proof. Consider the simplest case, a degree $2n = 4$ vertex, where we have four Kawasaki vertices, V_1, V_2, V_3, and V_4, and a vertex v which is on one edge off of each of the four given vertices.

Consider any single exterior vertex, say V_1 connected to two other exterior vertices (V_2 and V_4) and one interior vertex (v). Notice that angles $\angle V_2V_1V$ and $\angle VV_1V_4$ do not depend on the distance between the vertices, only on the direction from one to another.

No matter where V_2 and V_4 are, the 4th crease (in red) from V_1, $\overline{AV_1}$ can ALWAYS be chosen so that the V_1 is Kawasaki; simply measure the supplementary angle $\pi - \angle VV_1V_4$

counterclockwise from $\overline{V_2V_1}$ to form $\angle V_2V_1A$. Then the 4th angle ($\angle AV_1V_4$) will necessarily be $\pi - \angle V_2V_1V$ since the four angles add up to 2π. Choosing any other direction for the red ray will force V_1 to fail Kawasaki.

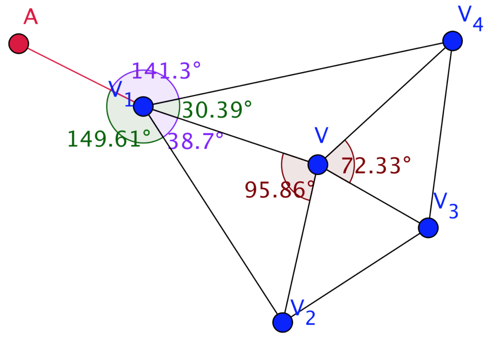


Figure 15. Given Four Kawasaki Vertices and an Inner Vertex

Whether or not the interior vertex V is Kawasaki has nothing to do with the direction of the red segment $\overline{AV_1}$. The "Kawasaki-ness" of the exterior vertices depends only on the relative positions of the three vertices it is connected to. In the first figure we see that V is clearly not Kawasaki since the sum of the alternate sector angles around it is 168.19°.

Continuing this for each of the four vertices, we get that while all four exterior vertices are Kawasaki, the interior vertex is not Kawasaki since the sums of the alternate sector angles around it are 168.19° and 191.81° which do not equal π.

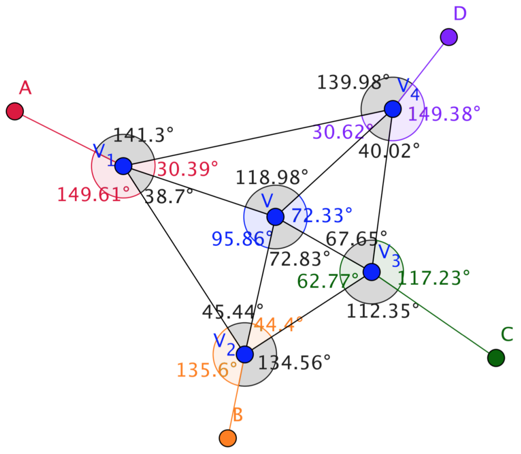


Figure 16. Given Four Kawasaki Vertices and an Inner Vertex

□

We have proved that given a $2n$-gon, an arbitrary point in the interior of the polygon need not necessarily Kawasaki.
Claim: Any point on a diagonal of a regular polygon, is a Kawasaki point.

Proof. Take a regular polygon P with $2n$ sides. Let X be a point on a diagonal of P. Join X to each of the $2n$ vertices of the polygon. This creates n sectors above the diagonal and n sectors below the diagonal, which are the mirror image of the sectors above. The sum of the sector angles below is π and the sum of the angle sectors is π. The sectors look like

$$\frac{\theta_1\ \theta_2\ \theta_3\ \ \ldots\ \ \theta_n}{\theta_1\ \theta_2\ \theta_3\ \ \ldots\ \ \theta_n},$$

where ———— is the diagonal containing X. Now, if we look at the sum of the alternate sectors around X, we have $\theta_1 + \theta_3 + \theta_5 + \ldots = \theta_2 + \theta_4 + \ldots = \pi$. □

Given a polygon, can we find the space of interior points that are Kawasaki? To rephrase, given the points of a polygon, where can we place an interior point so when it is connected to the points of the polygon, its sector angles are guaranteed to be Kawasaki? Since any Kawasaki point must have an even amount of sector angles, we only concern ourselves with even degree polygons.

Consider an arbitrary quadrilateral whose vertices are $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ as in the following figure.

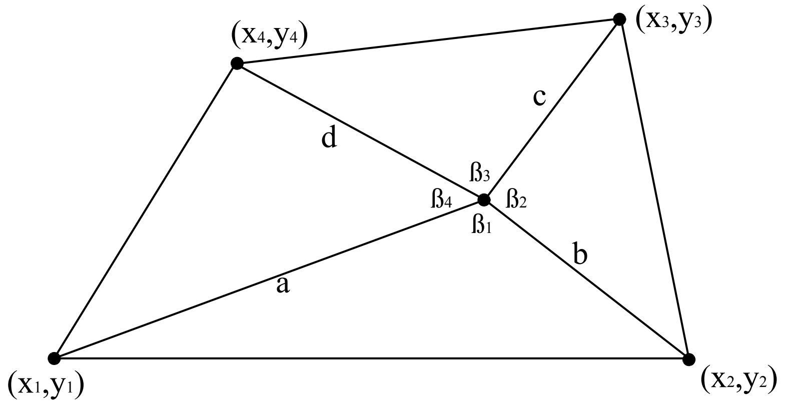


Figure 17. Arbitrary Quadrilateral

The goal here is to find an interior point (x, y) so that the β_i are Kawasaki. Let (x, y) be an arbitrary interior Kawasaki point. Then

$$\beta_1 + \beta_3 = \pi.$$

Consider the vectors a, b, c, d. Using the dot product we can express the sum of β_1 and β_3 in terms of a, b, c, d. Here

$$a \cdot b = |a||b|\cos(\beta_1),$$

hence

$$\arccos\left(\frac{a\cdot b}{|a||b|}\right)=\beta_1.$$

We can obtain a similar expression for β_3 using vectors c and d. Furthermore the sum $\beta_1+\beta_3=\pi$ can be written as

$$\arccos\left(\frac{a\cdot b}{|a||b|}\right)+\arccos\left(\frac{c\cdot d}{|c||d|}\right)=\pi. \tag{8.1}$$

Since we are given the exterior points and we have the interior point to be (x,y) we see that

$$a=\langle x_1-x,y_1-y\rangle,\;\; b=\langle x_2-x,y_2-y\rangle,\;\; c=\langle x_3-x,y_3-y\rangle,\;\; d=\langle x_4-x,y_4-y\rangle.$$

Using the above notation we write (8.1) as

$$\sum_{i=1}^{2}\arccos\left(\frac{\langle x_{2i-1}-x,y_{2i-1}-y\rangle\cdot\langle x_{2i}-x,y_{2i}-y\rangle}{|\langle x_{2i-1}-x,y_{2i-1}-y\rangle||\langle x_{2i}-x,y_{2i}-y\rangle|}\right)=\pi$$

which expands to

$$\sum_{i=1}^{2}\arccos\left(\frac{(x_{2i-1}-x)(x_{2i}-x)+(y_{2i-1}-y)(y_{2i}-y)}{\sqrt{[(x_{2i-1}-x)^2+(y_{2i-1}-y)^2][(x_{2i}-x)^2+(y_{2i}-y)^2]}}\right)=\pi.$$

These results generalize to a polygon of degree $2n$. If we label the vertices of a $2n$ polygon counter clockwise by $(x_1,y_1),(x_2,y_2),\ldots,(x_{2n},y_{2n})$ then the space of interior points $(x,y)\in\mathbb{R}^2$ that are Kawasaki satisfy the equation

$$\sum_{i=1}^{n}\arccos\left(\frac{(x_{2i-1}-x)(x_{2i}-x)+(y_{2i-1}-y)(y_{2i}-y)}{\sqrt{[(x_{2i-1}-x)^2+(y_{2i-1}-y)^2][(x_{2i}-x)^2+(y_{2i}-y)^2]}}\right)=\pi. \tag{8.2}$$

Note that in order for these points to be a part of the mesh, we can only take the points (x,y) which simultaneously satisfy (8.2) and lie in the interior of the $2n$-gon.

Definition 8.1. *The Kawasaki space of polygon P of degree $2n$ is the set of all solutions (x,y) to the equation (8.2) and lie inside the polygon P.*

We have a rough java program called "Squarasaki" which allows the user to input the vertices of a $2n$-gon and compute the Kawasaki curve. See the programs section for more information on this.

9. Programs

Title: Fold

Author: Jason Orozco

Language: Java

Purpose: To determine the number of folded states and mountain valley assignments for a given k-point.

How it works: The "Fold" program is based on the principle that every flat-folded vertex has a sector path that can be characterized by its angles together with a permutation of the numbers {1,2,...,2n}. This purely numerical characterization of flat-folded vertices made possible the design of a computer algorithm that generates and analyses them. To describe what the algorithm does mathematically, we provide the following definition of a "stack", which represents the angles of a vertex stacked on top of each other, and placed in a coordinate system.

Definition: A *degree* $2n$ *stack* for some natural number n is a $2n\times 2$ real matrix A together with a one-to-one function $p : \mathbb{N}_{2n} \longrightarrow \mathbb{N}_{2n}$ satisfying the following properties:

$$A = \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \\ \vdots & \vdots \\ a_{(2n)} & b_{(2n)} \end{bmatrix}$$

(*i*) *The origin property:* $a_{p(1)} = a_{p(2n)} = 0$.
(*ii*) *The segment property* : For all $i \in \mathbb{N}_{2n}$, we have $b_i > a_i$.
(*iii*) *The alignment property* : For all even k, $1 \leq k < 2n$, $a_{p(k)} = a_{p(k+1)}$. For all odd k, $1 \leq k < 2n$, $b_{p(k)} = b_{p(k+1)}$.

Connectibility Criteria: Let Ξ be a degree 2n stack for some natural number n, where $p(k)$, a_i, and b_j for $i, j, k \in \mathbb{N}_{2n}$, denote its components. For simplicity, let $p(2n + 1) := p(1)$. We call Ξ *connectable* if the following criteria are satisfied:

(*i*) For all even $k \in \mathbb{N}_{2n}$ and any i strictly between $p(k)$ and $p(k+1)$, either $b_i \leq a_{p(k)}$ or $a_{p(k)} \leq a_i$.
(*ii*) For all odd $k \in \mathbb{N}_{2n}$ and any i strictly between $p(k)$ and $p(k+1)$, either $b_i \leq b_{p(k)}$ or $b_{p(k)} \leq a_i$.
(*iii*) For all even $i, j \in \mathbb{N}_{2n}$ such that $i \neq j$ and $a_{p(i)} = a_{p(j)}$, either both or neither of $p(j)$ and $p(j + 1)$ are between $p(i)$ and $p(i + 1)$.
(*iv*) For all odd $i, j \in \mathbb{N}_{2n}$ such that $i \neq j$ and $b_{p(i)} = b_{p(j)}$, either both or neither of $p(j)$ and $p(j + 1)$ are between $p(i)$ and $p(i + 1)$.

Generating a stack from a k-point and permutation: We will demonstrate the process of constructing the stack's matrix with an example. Consider the k-point

$(50, 30, 30, 50, 60, 60)$ and permutation $p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 5 & 1 & 4 & 2 & 3 \end{pmatrix}$. It is helpful to read the permutation as "angle 1 goes in row 6, angle 2 goes in row 5, angle 3 goes in row 1,..." and so on, where "angle" refers to a term of the k-point. We can then write the stack as a matrix with an additional column augmented on the left for the permutation to be entered in that manner:

$$\left[\begin{array}{c|cc} 3 & a_1 & b_1 \\ 5 & a_2 & b_2 \\ 6 & a_3 & b_3 \\ 4 & a_4 & b_4 \\ 2 & a_5 & b_5 \\ 1 & a_6 & b_6 \end{array}\right]$$

The left column is the inverse of p when read from top to bottom. Next we set a_6 to 0. Since p(1) = 6, this satisfies the property $a_{p(1)} = 0$. Then we set b_6 to 50, the first term of the k-point. So far we have

$$\left[\begin{array}{c|cc} 3 & a_1 & b_1 \\ 5 & a_2 & b_2 \\ 6 & a_3 & b_3 \\ 4 & a_4 & b_4 \\ 2 & a_5 & b_5 \\ 1 & 0 & 50 \end{array}\right]$$

In accordance with the alignment property, we set b_5 to 50 as well, so that $b_{p(k)} = b_{p(k+1)}$ for $k = 1$. We set a_5 to $50 - 30 = 20$ so that the measure of angle 2 is the difference between a_5 and b_5

$$\left[\begin{array}{c|cc} 3 & a_1 & b_1 \\ 5 & a_2 & b_2 \\ 6 & a_3 & b_3 \\ 4 & a_4 & b_4 \\ 2 & 20 & 50 \\ 1 & 0 & 50 \end{array}\right]$$

Similarly, we set a_1 to 20 so that $a_{p(k)} = a_{p(k+1)}$ for $k = 2$. Then, set b_1 to $20 + 30 = 50$ so that the measure of angle 3 is the difference between a_1 and b_1.

$$\left[\begin{array}{c|cc} 3 & 20 & 50 \\ 5 & a_2 & b_2 \\ 6 & a_3 & b_3 \\ 4 & a_4 & b_4 \\ 2 & 20 & 50 \\ 1 & 0 & 50 \end{array}\right]$$

Continue this process for the remaining angles. The result is a stack satisfying the necessary properties, and for each angle i, its measure is preserved in the difference $b_i - a_i$.

$$\left[\begin{array}{c|cc} 3 & 20 & 50 \\ 5 & 0 & 60 \\ 6 & 0 & 60 \\ 4 & 0 & 50 \\ 2 & 20 & 50 \\ 1 & 0 & 50 \end{array}\right]$$

Once the stack's matrix is generated, the *connectibility criteria* can be checked using inequality comparisons of its entries. We consider a connectible stack to represent a valid flat-fold pattern for the vertex whose angles are the terms of the k-point used to generate the stack. An unconnectible stack does not represent a flat-fold pattern.

Counting Flat-fold Patterns:

A powerful result is that the set of all ways to flat-fold a vertex is given by the set of all connectible stacks that are generated by the vertex's angles together with different permutations. However, when counting the number of fold-patterns for a given vertex, it is desirable to first define equivalence relations on stacks that relate those which are in some sense the same pattern, based on different symmetries. We call such equivalence classes *folded-states*, and consider the number of folded-states to be the number of ways to flat-fold a vertex. It is important to note that the count achieved in this manner depends greatly on the strength of the equivalence relations used to partition the connectible stacks into folded-states.

Counting MV-Assignments:

Each stack has an MV-assignment associated with it that can be derived from the permutation used to generate the stack. We say that an MV-assignment x is valid for a particular vertex v if there is a connectible stack for v that x is associated with. Then we simply count the number of distinct MV-assignments that are each associated to some connectible stack.

The "Fold" Program:

The "Fold" program essentially does everything described above, then outputs the counting results as well as diagrams of each stack and their MV-assignments.

How to use it:

(a) Press 'ctrl+n' to input new angles or go to the "Input" menu on top and click "New Angles". A popup appears that says "Angles:". Enter an even number of integers from 2 to 10, separated by any non-number character and press OK. There is a cap on the angle size you can enter to prevent the program from crashing but it's ridiculously huge.

(b) Press 'a' to view all stacks or go to the "View" menu and select "Display All". The unconnectable stacks will be shaded.

(c) Press 'c' to view only connectable stacks or go to the "View" menu and select "Display Connectable".

(d) Press 'u' to view only unconnectable stacks or go to the "View" menu and select "Display Unconnectable".

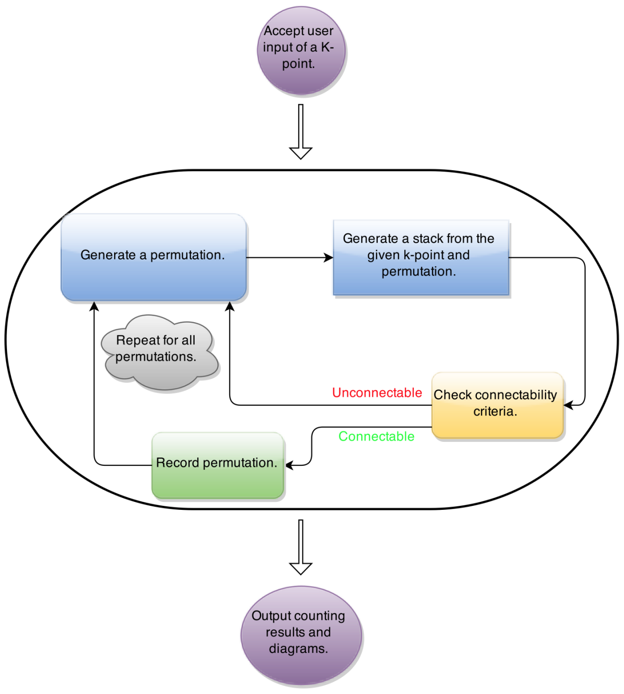
(e) Press 'v' to view the corresponding vertex inside a circle or go to the "View" menu and select "Vertex In Circle".

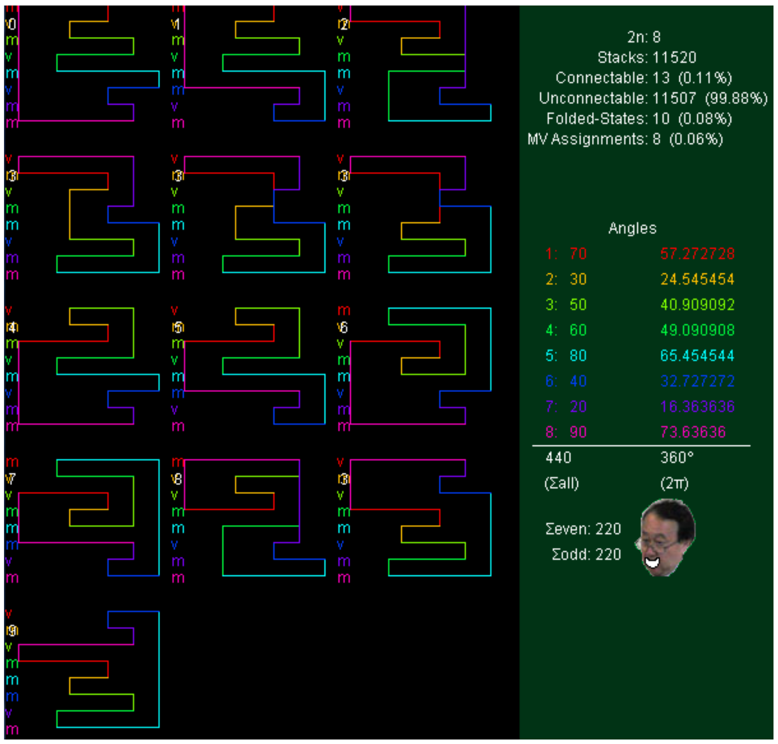
(f) Press 'b' to view the corresponding vertex inside a polygon or go to the "View" menu and select "Vertex In Polygon".

(g) Press 's' to save a screenshot of the viewing area without the info panel. There is no menu item for this. A popup appears asking for a filename without an extension. The program automaticlly attaches ".png" after you press OK. The image file will be saved in the Fold_v1 folder that the .jar file is in. This works in the stack views and the vertex views. The vertex views are intended to be nice for printing. The stack views are colored nicely to be viewed on-screen, not printed.

(h) Press 'up' and 'down' to stretch the stack diagrams vertically.

Screen-shot:


Accept user input of a K-point.
Generate a permutation.
Generate a stack from the given k-point and permutation.
Repeat for all permutations.
Unconnectable
Check connectability criteria.
Connectable
Record permutation.
Output counting results and diagrams.



```
2n: 8
Stacks: 11520
Connectable: 13  (0.11%)
Unconnectable: 11507  (99.88%)
Folded-States: 10  (0.08%)
MV Assignments: 8  (0.06%)

Angles
1:  70          57.272728
2:  30          24.545454
3:  50          40.909092
4:  60          49.090908
5:  80          65.454544
6:  40          32.727272
7:  20          16.363636
8:  90          73.63636
440             360°
(Σall)          (2π)

Σeven: 220
Σodd: 220
```

Title: Random k-point generator

Author: Lucas Mattick

Language: Python

Purpose: To randomly generate k-points of a given degree and alternate sum. In particular: To uniformly generate points $(x_1, x_2, \ldots, x_n)$ so that

$$x_1 + x_3 + \cdots + x_{2n-1} = x_2 + x_4 + \cdots + x_{2n} = r$$

for some $r, n \in \mathbb{Z}_+$.

How it works: Given the degree (n) and alternate sum (r) of the desired k-point, first it generates $n-1$ *distinct* random integers $\{z_1, z_2, \ldots, z_{n-1}\}$ in the interval $(0, r)$ using the random number generator in python. These numbers can be thought of as where we divide up the number line as in Figure 18.

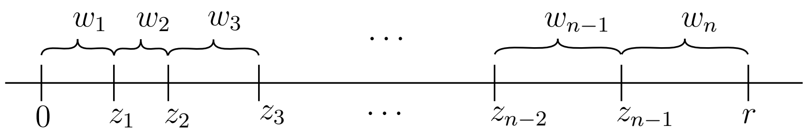


FIGURE 18.

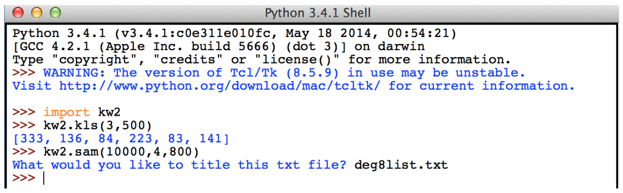
With these $n-1$ numbers, we obtain n numbers $\{w_1, w_2, \ldots, w_n\}$ whose sum is r by construction. Take

$$w_1 = z_1,\ w_2 = z_2 - z_1,\ w_3 = z_3 - z_2, \ldots,\ w_n = r - z_{n-1}$$

Repeat the process one more time to obtain a second set of n numbers whose sum is r. Take the first set of n numbers to be $\{x_1, x_3, \ldots, x_{2n-1}\}$ and the second set of n numbers to be $\{x_2, x_4, \ldots, x_{2n}\}$. Then we have the desired point $(x_1, x_2, \ldots, x_{2n})$ so that $x_1 + x_3 + \cdots + x_{2n-1} = x_2 + x_4 + \cdots + x_{2n} = r$.

How to use it: Open up a new python shell and type `inport kw2`. With this you can either generate one k-point, or a list of k-points. To generate one k-point type in `kw2.kls(n,r)`, this will generate one k-point of degree $2n$ with $\sigma = r$. The k-point should output to the current shell. To generate a list of k-points type in `kw2.sam(m,n,r)`, this will generate a list of m k-points of degree $2n$ with $\sigma = r$. The shell will prompt you with,"What would you like to name this file?" Name the file however you like with the extension `.txt`. The file should appear in the same folder that you have `kw2.py` saved.

Screen-shot:



```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> import kw2
>>> kw2.kls(3,500)
[333, 136, 84, 223, 83, 141]
>>> kw2.sam(10000,4,800)
What would you like to title this txt file? deg8list.txt
>>> |
```

Title: The Meandric Permutation Solver
Author: Rishidhar Reddy Bommu
Language: Java
Purpose: The program generates meandric permutations of degree $2n$ for a given n.
How it works: **Definition:** A meander is a self-avoiding closed curve that intersects a line a number of times.

Definition: A permutation of the set $\{1, 2, ..., 2n\}$ starting with 1 determined by a meander is a meandric permutation.

To obtain all meandric permutations of degree $2n$ we need to generate all closed meanders of degree $2n$ first. To do this we break a meander into its upper and lower half with respect to the line it is intersecting. In Figure 19, the up and down arrows represent where the meander enters and exits the upper half. Notice that for every up arrow there is a corresponding down arrow. We can represent the up and down arrows as left and right parentheses respectively. In doing this we are left with a balanced set of parentheses. Similarly we do this for the bottom half. Now we can represent the meander in Figure 19 as a pair of balanced sets of parentheses.

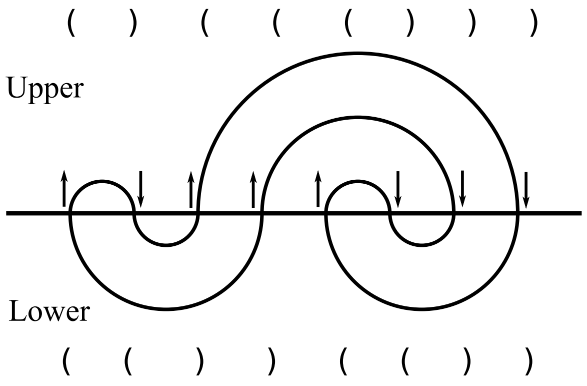


Figure 19. A Meander of Degree 8

Using this method we generate all balanced sets of parentheses for a given n. This represents all the possible forms the upper and lower half of a meander can take. With these we can compare different pairs of balanced sets of parentheses to obtain all meanders for a given n, thus giving us the respective meandric permutations.

Meandric Permutation Algorithm explanation

Step 1: Compute all different combinations of a set of $2n$ Parentheses with the following conditions.
CONDITIONS:
(i) Each left parenthesis has a matching right parenthesis.
(ii) Matched pairs of parentheses are well nested.

Step 2: Store all possible balanced sets of parentheses in a stack and copy that stack. Let us call them stack 1 and stack 2. These two stacks represent our possible upper and lower halves of the meander to be compared.

Step 3: Compare every element in stack 1 to every element in stack 2. when compared if the two balanced sets of parentheses form a single closed curve, we know it represents a closed meander.

Step 4: Store all closed meanders.

Step 5: Generate all meandric permutations from the closed meanders.

Meandric Algorithm Pseudo code

<u>Balanced Sets of Parentheses:</u> Determine all balanced sets of parentheses for a given n. Compare the balanced sets of parentheses to themselves to determine all closed meanders.

<u>Comparing Parentheses:</u> If the pair of balanced sets of parentheses represents a single curve that intersects the line $2n$ times, then we have a closed meander.

<u>Meandric Permutations:</u> For every closed meander generate its respective meandric permutation.

<u>How to use it:</u> The program will Initially request the user to select one of four programs. All four programs run the same way, but will output different results and provide different speeds. Then the user is prompted to enter a n. Then the program will spend time computing the results.The output will be written to results.txt(results may vary depending on user request).

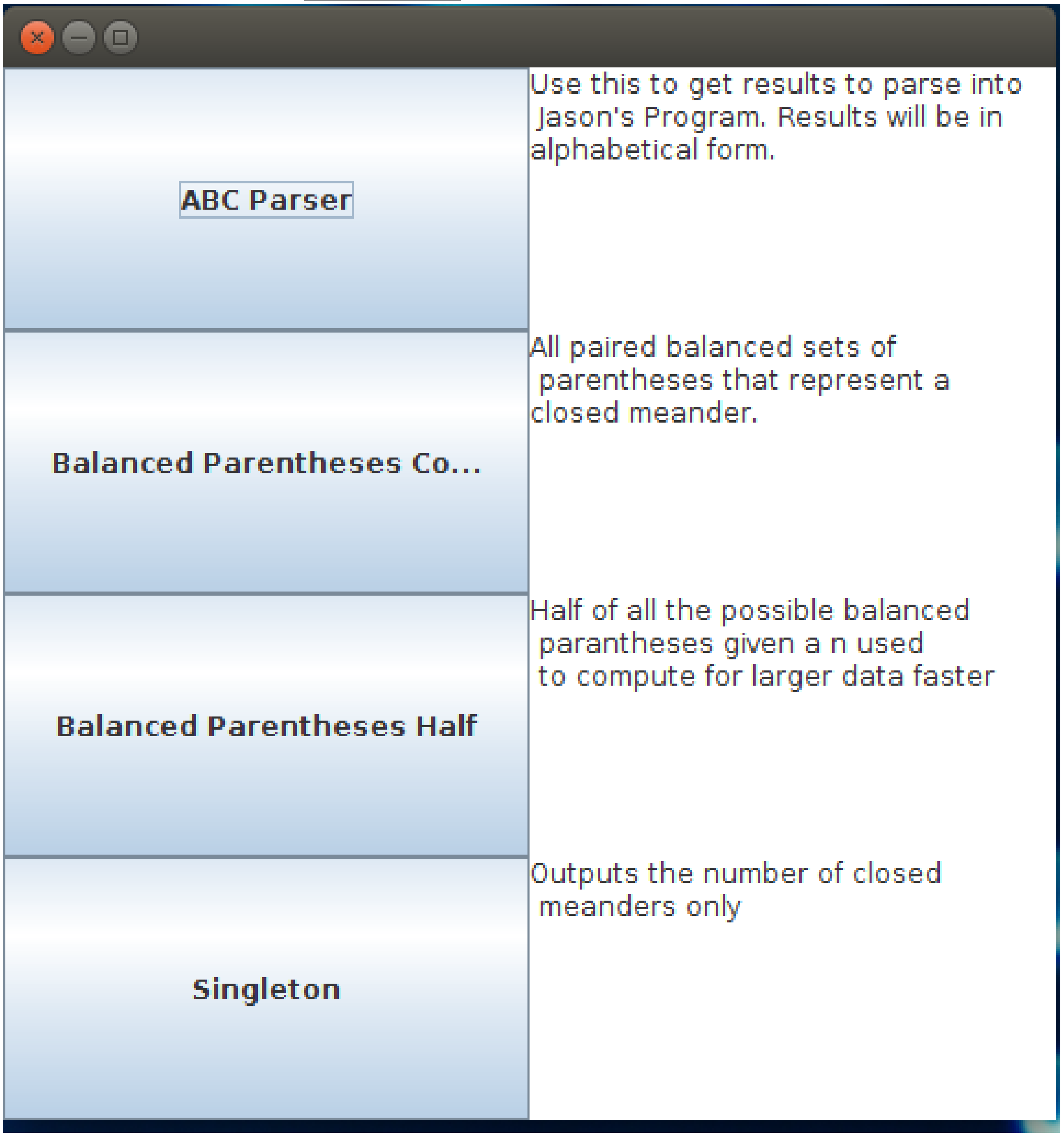
Program 1 ABC Parser- This program will spend an extra step converting all the Meandric Permutations which are represented in numeric form to a alphabetical form. The output is still the same, but in terms of letters instead of numbers. For example, 1 is now A; 2 is now B; and so forth. The cycle notation is still present and preserved.

Program 2 Balanced Parentheses Complete- This will output the complete list of Meandric Permutations and Balanced Set of Parentheses.

Program 3 Balanced Parentheses Half- This will output only half the results of the previous program by choosing to execute and output only uniques sets of Balanced Parentheses. For example, given two identical stacks(upper and lower) comparing element 4(stack 1) to element 2(stack 2) gives the same results as comparing element 2(stack 1) to element 4(stack 2). Thus the program in this version will only compute the first time and output it, but not the second. This will result in a much faster program, but outputs less results.

Program 4 Singleton- This program in terms of design will only keep a single instance of the count in the program. It will use the least amount of memory and will show results the fastest. However as a con, it will only output the number of meanders there were and nothing else.

Screen-shot:



Use this to get results to parse into Jason's Program. Results will be in alphabetical form.

All paired balanced sets of parentheses that represent a closed meander.

Half of all the possible balanced parantheses given a n used to compute for larger data faster

Outputs the number of closed meanders only

10. Appendix

Degree 10:

# of MV Assignments	# of Regular K-Points
16	12581
24	15433
32	744
36	7672
40	359
48	699
54	1775
60	324
70	10
72	266
80	14
81	190
90	99
100	6
105	9
108	43
112	1
120	11
126	1
135	10
140	1
150	1
168	1
210	1

Folded States for Degree 10

# of Folded States	# of Regular K-Points	# of Folded States	# of Regular K-Points
16	759	67	114
17	50	68	280
18	440	69	216
19	91	70	350
20	967	71	126
21	150	72	400
22	720	73	103
23	229	74	166
24	2198	75	128
25	240	76	266
26	1120	77	146
27	308	78	317
28	1339	79	114
29	344	80	157
30	1293	81	138
31	327	82	158
32	1908	83	72
33	413	84	241
34	1326	85	86
35	496	86	165
36	1909	87	69
37	458	88	130
38	1050	89	41
39	505	90	95
40	1217	91	82
41	328	92	167
42	836	93	61
43	357	94	100
44	1318	95	41
45	327	96	123
46	890	97	39
47	315	98	64
48	1222	99	55
49	244	100	115
50	744	101	40
51	354	102	68
52	660	103	34
53	326	104	61
54	540	105	93

# of Folded States	# of Regular K-Points	# of Folded States	# of Regular K-Points
55	311	106	113
56	843	107	41
57	372	108	52
58	497	109	16
59	262	110	31
60	602	111	47
61	217	112	54
62	481	113	20
63	308	114	152
64	387	115	29
65	174	116	19
66	415	117	8
118	35	169	6
119	45	170	10
120	37	171	3
121	22	172	4
122	25	173	2
123	30	174	5
124	31	175	2
125	26	176	3
126	31	177	1
127	18	178	6
128	31	179	2
129	31	180	9
130	46	181	8
131	14	182	2
132	34	183	6
133	6	184	5
134	14	186	12
135	17	187	17
136	24	188	4
137	32	189	1
138	18	191	1
139	9	192	9
140	11	193	8
141	28	194	4
142	37	195	1
143	14	196	6
144	14	197	1
145	2	198	1

# of Folded States	# of Regular K-Points	# of Folded States	# of Regular K-Points
146	10	199	1
147	16	200	14
148	12	201	3
149	10	202	4
150	14	203	2
151	6	204	2
152	5	205	2
153	8	207	3
154	27	208	4
155	16	210	3
156	16	211	4
157	4	212	6
158	8	214	4
159	2	216	1
160	4	217	3
161	14	219	3
162	22	222	3
163	11	224	1
164	6	225	6
165	26	227	2
166	5	230	5
167	4	232	3
168	10	233	1
236	4	416	2
240	3	418	2
241	1	440	1
243	1	442	1
244	4	448	1
245	2	456	1
247	2	485	1
249	2	538	1
250	2	540	1
251	3	729	1
252	1	1310	1
253	1		
254	2		
256	3		
257	6		
262	3		
263	2		

# of Folded States	# of Regular K-Points	# of Folded States	# of Regular K-Points
264	2		
266	2		
267	2		
268	1		
269	2		
272	2		
273	1		
275	1		
285	2		
288	1		
289	2		
290	2		
294	1		
296	2		
297	1		
299	1		
303	1		
305	1		
306	2		
309	2		
311	2		
314	1		
315	1		
320	1		
321	1		
336	1		
344	1		
345	1		
353	4		
359	1		
370	1		
377	1		
404	1		
412	1		