展開図に基づく折りたたみ方の数え上げ

Counting the Number of Unique Configurations of Flat-Folded Origami Reconstructed from a Crease Pattern

三谷 純
Mitani Jun

はじめに

折紙作品とその展開図の関係は極めて密接で、複雑な折紙作品の創作には展開図に対する深い考察が必要不可欠なものになっています。最近では与えられた展開図だけを手がかりに、そこから形を折りあげる「展開図折り」も多くの方に親しまれているようです。

折紙作品の形が決まっているのであれば、それを開いて得られる展開図は言うまでもなく 1 通りに定まります。では､その逆はどうでしょう｡つまり、1 つの展開図が与えられた場合に、そこから折りあげられる作品は 1 通りに決まるでしょうか。もし 1 通りに決まらないとしたら、可能な折り方を全て数え上げるにはどうしたらよいでしょうか。この問いに対して、コンピュータを使って答えを見つけよう、というのが今回のテーマです。

形態数の数え上げ問題

ここでは対象とする折り方を平坦折りとし、折り線には予め山谷の区別がつけられているものとします。まず簡単な例を取り上げることで、問題をわかりやすくまとめてみましょう。

図 1 左の展開図は、誰が折っても右側に示す 1 通りにしか折ることができません（以降では、異なる折り方の数を「形態数」と呼ぶこととし、図 1 左の展開図に対して｢形態数は 1 である｣と言うことにします)。

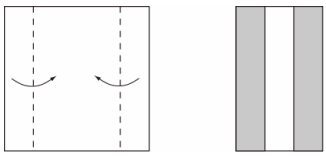

図 1. 形態数 1 の展開図とその折り方

それでは、図 2 左の例ではどうでしょう。2 本の谷折線がありますが、どちらを先に折るかによって、右側に示すように 2 つの異なる結果となります。従って図 2 左の展開図の形態数は 2 です。

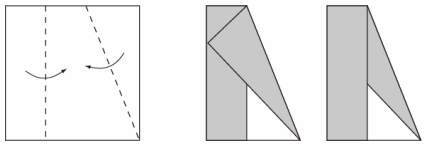

図 2. 形態数 2 の展開図とその折り方

このような簡単な例では、実際に折って形態数を数え上げることが可能ですが、複雑な例になると数え漏れが出てくるかもしれません。そこで、コンピュータを使って全ての形態数を正確に数え上げることを試みました。その方法と結果を以降で紹介します。

コンピュータを用いた数え上げ手法

今回、形態数を数えあげるためにコンピュータに実装した処理の手順を次の(1)～(4)に示します（以降の説明で「面」とは折り線で囲まれた領域を指すこととします)。

(1) 山谷の区別がついた展開図データを用意する。
(2) 展開図を折った後の面の位置と向き（表面が上を向いているか下を向いているか）を求める。
(3) 1～N（N=面の数）の重なり順を各面に重複無く割り当てたときに、展開図の山谷条件と矛盾しないものを全て列挙する（1 が割り当てられた面が一番下に、N が割り当てられた面が一番上に配置されるものとする)。
(4) (3)で列挙された順番で面を重ねたときに、同じ形態となるもの除いて、ユニークな形態をのみを数え上げる。

以降では、各手順の詳細を説明します。

手順(2)は、展開図に含まれる頂点が「局所平坦条件」を満たしていれば次の方法で容易に求めることができます。

> **展開図を平坦に折りたたんだ時の形状の求め方**:ある面を開始面として隣接する面を巡回する。その際に、折線を跨ぐたびに、その折線で隣接面を山折りなら 180 度、谷折りなら-180 度回転させる。

この処理は Web 上で一般公開している ORIPA (http://mitani.cs.tsukuba.ac.jp/pukiwiki-oripa/)というソフトウェアにも実装されており、鶴の展開図に適用した結果は図 3 のようになります。折りたたまれた後の形を正しく得られることを確認できます。ただし、この時点では面の重なり順は求まっていません。

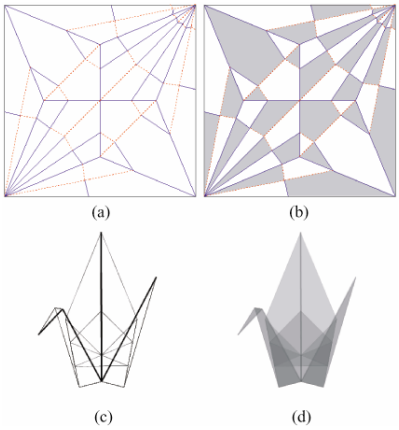

図 3　(a)展開図、(b)面の向きによって色分けされた展開図、(c)折りたたみ後の面の位置、(d)透過色で塗りつぶしを行った様子

手順(3)では、N 個の面に 1～N の重なり順を割り当て、展開図から得られる情報と矛盾するものを省きます。調べるべき順番の割り当て方は単純に考えると N!の場合の数があり、これは面の数が多くなると爆発的に大きな数になってしまうことを意味します。たとえば、面の数が 18 の兜（図 8(c)）の例では $18!=6.4\times10^{15}$ となり、コンピュータを使ってもとても調べ切れません。

そこで、面を（スタックに）下から順番に積み重ねていきながら、矛盾が生じた時点で積み重ねを終了し次のケースに移行する、という処理を行うことで、無駄な処理を効率的に枝刈りし、調べあげる対象の数を大幅に減らすことを行います。「矛盾が生じる」というのは、例えば図 4 のように、上の面が下の面の折線を覆い、正しく折りたためないようなケースを言います。

図 4. 折紙の断面を見たところ（点線は面のつながりを表す）。面 F_1 の折線 e は上の面とつながるべきだが、その上に面 F_2 が覆いかぶさっている。

折り紙を扱う上で非常に厄介な問題として、複数の折り線が同じ位置に重なる場合の扱いがあります。図 5 のように一箇所に折り線が重なるケースでは、F_2 と F_3 が接続する(a)はよくても F_1 と F_3 が接続する(b)は問題となります。紙幅の都合で詳しくは紹介できませんが、このような場合でも F_1 から順に面を重ねながら、面の向きと折線の山谷、および折線を介して接続する面の位置をチェックすることで、F_3 まで重ねた時点で面の干渉（(b)のケース）を発見することができます。

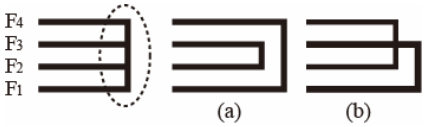

図 5. 折線が一箇所に集中する場合、面の重なり順によって正しく折れる場合(a)とそうでない場合(b)がある。

さて、以上に述べた方法で展開図と矛盾の無い重なり順を全て求めることができます。

ところが、図 6 のようなケースでは、異なる重なり順の割り当て方、$F_1<F_2<F_3$ と $F_1<F_3<F_2$ が同じ形態を表します。

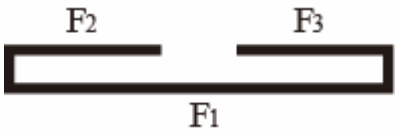

図 6. 得られる重なり順が 2 つ、形態数が 1 の例

異なる重なり順が、同じ形態を表す理由として、互いに干渉する位置に無い面（図 6 では F_2 と F_3）の存在が挙げられます。互いに干渉しない（離れた位置にある）面にとっては、どちらの重なり順が上でも折りたたまれた結果には影響を与えないのです。

そこで手順(4)によって、重複するものを除き、ユニークな形態だけを数え上げます。形態は（少しわかりにくい表現かもしれませんが）「干渉する位置関係にある 2 面間の上下関係の集合」で表すことができます。このような関係を表すには行列を用いると便利です。そこで、以下に述べる処理を行います。

まず、折りたたんだ後の 2 次元平面において、2 つの面が互いに干渉するか否かを表す N×N 行列 M_{mask} を作成します。M_{mask} の各要素の値 m_{ij} は次のように定めます。

$$m_{ij}=\begin{cases}1 & F_i \text{と} F_j \text{が互い干渉する} (i\neq j)\\ 0 & \text{それ以外}\end{cases}$$

次に、手順(3)で得られた重なり順に基づいて、面間の上下関係を行列 M_{order} で表現します。M_{order} の要素 m_{ij} は次のように定めます。

$$m_{ij}=\begin{cases}1 & (Order(F_i)>Order(F_j))\\ 0 & (i=j)\\ -1 & (Order(F_i)<Order(F_j))\end{cases}$$

Order(F)は面 F に割り当てられた重なり順で、$Order(F_i)>Order(F_j)$ であれば、面 F_i が面 F_j より上の番号を持っていることを意味します。続いて、次のような成分同士のかけ算を行う演算子⊗を定義します。

$C=A\otimes B$ の(i, j)成分 $c_{ij}=a_{ij}\,b_{ij}$

以上の定義に基づき、$M_{mask}\otimes M_{order}$ を行うと、互いに干渉する位置にある面同士の上下関係だけを表した行列（つまり 1 つの形態を表した行列）が得られます。従って手順(3)で得られる解を表すそれぞれの M_{order} に対して $M_{mask}\otimes M_{order}$ を求め、ユニークな解の数を数え上げればよいことになります。

数え上げの具体例

先に述べた手順(4)の説明を改めて具体例を元に説明します。対象として図 7 の折りたたみ方を考えることにしましょう。

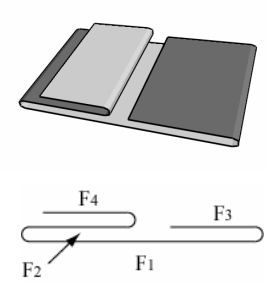

図 7. 面が4つの例題

まず手順(2)で折りたたみ後の各面の位置を算出し、面の位置から互いに干渉する面の対を調べます。図 7 の例では、F_1 と$\{F_2,F_3,F_4\}$が重なっており、F_2 と$\{F_1,F_4\}$、F_3と$\{F_1\}$、F_4と$\{F_1,F_2\}$が重なっているので、M_{mask}は次のようになります。

$$\mathbf{M}_{mask} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

手順(3)で述べた総当たりの方法で得られる重なり順の解は以下の 3 通りです。

解[1]: $F_1<F_3<F_2<F_4$

解[2]: $F_1<F_2<F_3<F_4$

解[3]: $F_1<F_2<F_4<F_3$

これらの解のそれぞれの行列 M_{order}は次のように表されます。

$$\mathbf{M}^{[1]}_{order} = \begin{pmatrix} 0 & -1 & -1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & -1 & 0 & -1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\mathbf{M}^{[2]}_{order} = \begin{pmatrix} 0 & -1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ 1 & 1 & 0 & -1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\mathbf{M}^{[3]}_{order} = \begin{pmatrix} 0 & -1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & -1 & 0 \end{pmatrix}$$

それぞれの行列に対し、M_{mask}との演算⊗を施すと、次のようにすべてが同じものになります。

$$\begin{aligned} &\mathbf{M}^{[1]}_{order} \otimes \mathbf{M}_{mask} \\ &= \mathbf{M}^{[2]}_{order} \otimes \mathbf{M}_{mask} \\ &= \mathbf{M}^{[3]}_{order} \otimes \mathbf{M}_{mask} \\ &= \begin{pmatrix} 0 & -1 & -1 & -1 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \end{aligned}$$

このとことから、図 7 に示す展開図の形態数は 1 であることがわかります。

実験結果

今回提案した手法を実際に図 8 に示す 4 つの展開図に対して実行した結果は表 1 のようになりました。

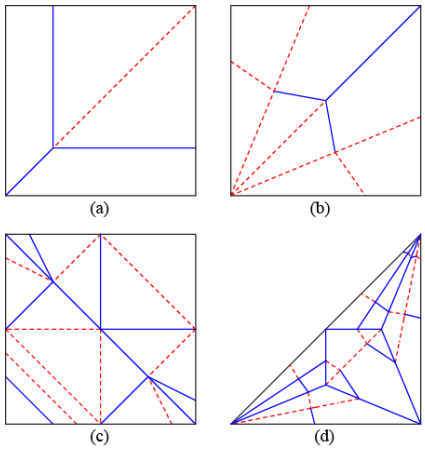

図 8. 例題。(a)(b)は単純な例、(c)は兜、(d)は鶴の展開図の半分

	N	N!	#Ans	#Ans'	time
(a)	4	24	1	1	0.22
(b)	8	$4.0\text{x}10^4$	3	3	0.31
(c)	18	$6.4\text{x}10^{15}$	2778	9	1.0
(d)	26	$4.0\text{x}10^{26}$	$5.0\text{x}10^5$	1	110

表 1. 図 8 の例題に適用した結果

表中の N は面の数、#Ans は手順(3)で得られた解の数、#Ans'は手順(4)で最終的に得られた形態数、time は計算に要した時間（秒単位）です。

単純な例題である図 7(b)については、一見すると中割り折りであると判断できますが、他に 2 通りの折り方があることが判明しました。兜の例では、4 万近くの妥当な重なり順が求まりましたが、最終的には 9 つの形態が見つかりました。この値が果たして正しいものであるか、試しに折ってみたところ、図 9 に示すように、確かに 9 通りあることが確認できました。

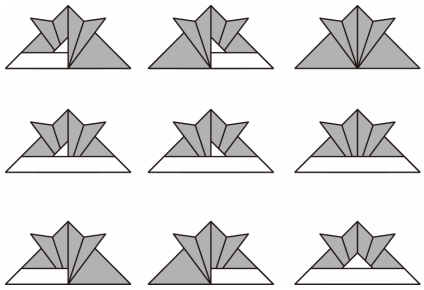

図 9. 例題。(a)(b)は単純な例、(c)は兜、(d)は鶴の展開図の半分

鶴の半分の展開図では、ただ 1 通りしかないことがわかりました。

おわりに

鶴の半分の展開図では、ただ 1 通りしかないことがわかりましたが、では鶴の全体の展開図ではどうでしょうか。実は、実験を行ってみたのですが、今回提案した手法でも、調べるべき場合の数が爆発的に増えてしまい、丸 3 日間コンピュータに計算させ続けましたが、残念ながら調べきれませんでした。ここからは、私の手による実験結果ですが、図 10 の丸で囲った領域を尾（脚？）の部分に差し込むことができることがわかりました。図の手前側と奥側でそれぞれ 2 通りの差し込み方があるので、通常の鶴とあわせて 5 通りの形態数があることがわかります。ところが尾の立ち上がり方が急な図 10 の右側の例の場合、このような差し込みができないので、形態数はただ 1 つと言うことができそうです。尾の角度によって、得られる形態数が異なる、という興味深い結論が得られました。

是非、実際に折って試してみてください。

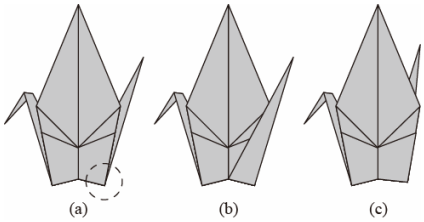

図 10. (a), (b)鶴の展開図から得られる異なる形態 (c) 形態数が 1 になる場合

鶴の展開図の形態数については、最終的に人の手によって得られたものですが、実このような解は、プログラムをデバッグしているときに発見したものなので、コンピュータを用いることが役立ったとも言うことができます。

今後は、今回紹介したアルゴリズムを改良し、面の数が多い複雑な展開図でも、実用的な時間で答えを求められるようにしたいと考えています。