展開図から推定される折りたたみ形態の数え上げ手法

三谷 純 Jun MITANI

概要: 対象を平坦に折りたたまれる折紙に限定することで、山折、谷折および輪郭線から構成される展開図から、その折紙が折りたたまれた状態の外形を決定できる。しかし、折紙の構成面素(折り線に囲まれた多角形)がどのような重なり順になるかを推定することは、構成面素同士の干渉を考慮しなければならないため難しい問題である。本稿では、計算機内に構成面素を重なり順に格納するスタックを用意し、実現可能な重なり順を総当たりで調べ上げる手法を提案する。さらに、実現可能な面の重なり順から、折紙作品として見たときに、同一な解を除き、異なる作品となる場合の数だけを数え上げる手法を提案する。

キーワード: 平面幾何学／折紙／展開図／数え上げ

1. はじめに

「折紙」は世界的にも広く認知され、学術的な研究も多くされている。特に最近では計算機環境の発達と普及により、折紙を計算機で扱う研究も多い。また近年になって作品の創作に「設計」の概念が持ち込まれるようになり、展開図から折紙の形を考察することが必要不可欠となっている。

本稿では、折紙の「展開図(図 2(a))」から折紙の完成形(展開図上の折り線に矛盾無く折りたたんだ後の形状)を計算機で推定する手法を提案する。なお、本稿で扱う展開図は山折と谷折の区別がされているものとし、さらに折りたたみ後の形が平坦になるもののみを対象とする。このように対象を限定することで、展開図から折紙が折りたたまれた状態の外形を決定できる(4章)。しかし、紙がどのような重なり順になるかを推定することは、構成面素同士の干渉を考慮しなければならず、また 1 つの展開図には複数の解が存在しうするため、どのように全ての解を漏れなく数え上げるかは難しい問題である。重なり順を決定する問題は、BernらによりNP困難であることが示されており^[1]、その論文の中では 1 つの展開図から異なる折りたたみ順がいくつあるかを決定する問題が未解決問題として挙げられている。本稿では、計算機を用いて面を下から順にスタックに格納した場合の格納順(スタッキングオ

ーダー)を決定することで、面の重なり順を決定する手法を提案する。ところで、スタッキングオーダーが異なっても、折紙の作品として見たときに同じ形になる場合があるため、後処理として得られた解からユニークな解を数え上げる必要がある。本稿では、この数え上げの手法も提案する。

これらをまとめると、展開図から折りたたみ後の形状を推定するために、本研究では以下の 3 つの問題に対する解法を示す。

- (1) 展開図を折りたたんだ後の構成面素の配置(2次元上の位置)を決定する問題
- (2) 実現可能なスタッキングオーダーを算出する問題
- (3) 得られたスタッキングオーダーからユニークな解を数え上げる問題

本稿では、これらの 3 つの問題それぞれについての解法を提案する。またそれらの手法を実際に計算機上に実装して実験を行った。

第 2 章では関連する研究を紹介し、第 3 章で対象とする展開図について述べる。第 4,5,6 章では上記(1),(2),(3)それぞれの問題を対象とした解法を述べ計算機に実装した結果を示す。第 7 章でまとめと今後の展望を述べる。

2. 関連研究

折紙は正方形の紙に対して折り操作を繰り返すことで様々な形を作るものであり、幾何の分野における研究題材として多く取り上げられている^{[2][3]}。また、国内に留まらず海外でも折紙の設計手法についての研究が行われており、近年では対象とする形状を折り出すための「設計」の概念が導入され、複雑な形状を持つ作品が多く作られるようになった^[4]。文献[3]で紹介されている折紙の設計においては、完成形の基本構造に基づいて展開図上に木構造を配置することを行うため、折紙の展開図とその折りたたみ後の形状に対する考察が必要不可欠である。

展開図を計算機で扱う研究として、内田ら^[5]は展開図を構成する幾何学的要素から折りたたみ方法を推論するプログラムを提案した。この研究は本稿の扱う研究テーマと非常に近いが、対象とする展開図を構成する折り線が「山折線」と「谷折線」だけ

でなく、「山折に折って開い線」と「谷折に折って開い線」も含まれる点が本稿の対象と異なる。つまり内田らの対象とする展開図には、最終的な形を決定するのに用いられない折り線も含まれ、本稿が対象とする展開図よりも情報が多い。

Hull は山折、谷折の区別のない展開図から、折り線の山折り、谷折りの違いを決定する問題について言及している^[6]。本研究では、予め山折りと谷折りの違いが明らかであることを前提とする。

島貫ら^{[7][8]}は、折り操作によって得られる妥当な面の重なり順の算出方法を提案しているが、これは特定の切断面に着目して推定しているため、局所的な領域の折りたたみ順を求めることはできるが、全体の面の重なり順を漏れなく数え上げる目的にはそのままでは使用できない。

3. 入力に用いる展開図

折紙の展開図は「山折線」、「谷折線」および紙の輪郭を表す「輪郭線」の 3 種類の線分の集合から構成される。これらの線分を効率的に入力するためのエディタとして、既に開発済みの ORIPA (ORigami PAttern editor)^[9]を用いた。

図 1 ORIPA の画面

与えられた展開図が平坦に折りたためるために必要十分な、展開図の内点に接続している折れ線の条件(局所平坦条件)は文献[10][11][12]より、次の通りである。

- (1) 折れ線の数 は偶数である
- (2) 山折線の数と谷折線の数 の差の絶対値は 2 である
- (3) 折れ線の成す角のひとつおきの和は 180° である
- (4) 折れ線の成す角が鈍角の時、その角をはさむ 2 つの線の折れ線属性(山折/谷折)は等しい

これらの条件を全ての内点が満たす展開図は平坦に折りたたむことができる。本研究では、入力された展開図は、上記の条件を満たすことを前提としている。

4. 展開図の折りたたみ形状の推定

展開図内の折れ線情報に基づき、計算機内で紙の折りたたみを行い、折りたたみ後の形状を構築する。この問題は、島貫ら^[5]によっても実装されており、とくに新しい手法ではないが、本研究で用いた手法を以下に示す。

まず、折り線または輪郭線によって囲まれた閉領域(以降、多角形面素、または単に面と呼ぶ)が折りたたみ平面上でどの位置に移動するかを推定する。はじめに起点となる任意の多角形面素を決定し、それに隣接する多角形面素を再帰的に巡回する。隣接面に移動する毎に、2 面間を介する稜線によって(山折、谷折の別に依らずに)多角形面素を折り返す(つまり平面上で折り線を介して反転させる)ことを行う。この反転操作は、各折り線線分に対して高々 1 回行うものとする。これにより、基点となる面から奇数回の移動で到達する場合は面の向きが裏に、偶数回の移動で到達する場合は面の向きが表になる。全ての多角形面素を巡回した時点で折りたたみ操作が完了する。

上記の手法で、展開図から多角形面素を取得し、その折りたたみ後の位置を算出することで図 4 の結果が得られた。図(a)は鶴の展開図であり、(b)は折りたたみ後に表面が上を向く多角形面素を網掛けで表している。(c)は折りたたみ後の多角形面素の輪郭を表示したもの、(d)は透過色で塗りつぶしたものである。これにより、面の向きが隣接する多角形面素で互いに異なること、及び前述の方法で妥当な折りたたみ後の形状を得られることがわかる。なお、この時点では多角形面素の重なり順は不定である。

図 2 (a)展開図、(b)面の向きによって色分けされた展開図、(c)多角形面素の輪郭、(d)透過色で塗りつぶした多角形面素

5. スタッキングオーダーの算出

前章で述べた折りたたみ操作で、平面上での各多角形面素の位置を得ることができるが、それぞれの重なり順は求まらない。以降では、各多角形面素の重なり順を決定する方法を紹介する。

5.1. 多角形面素の重なり順の制約

多角形面素の重なり順には制約があり、折り線に従って折り操作を行う場合、多角形面素どうしが衝突して実現できない重なり順が存在する。例えば、図5を例に挙げると、 F_1 を最下層に配置した場合(a)の展開図では F_0 と F_2 のどちらが上でも問題ないが、(b)は F_0 が F_2 の下になることができない。この例から、多角形面素の重なり順を決定するには、展開図の位相情報からだけでは判断できず、多角形面素の形状に基づく幾何的な判定が必要があることがわかる。

図3 多角形面素の重なり順の制約(点線は谷折)

5.2. 妥当な重なり順の算出

面の重なり順を決定する問題は Bern らにより NP 困難な問題であることが示されている^[12]ため、提案手法では取り得る全ての重なり順を対象に探索を行い、折紙として妥当な重なり順であるものを抽出することとする。ところで、多角形面素の数を N とすると、その重ね順の場合の数は $N!$ 通りあり、 N が増大するとすぐに場合の数は爆発してしまう。例えば図2の鶴の例では多角形面要素の数が52であり、 $52! = 10^{67}$ 通りの重なり順を全て調べるのは現実的でない。そこで、できるだけ判定の数を減らすために図4に示すアルゴリズムを用いる。

このアルゴリズムでは、始めに空のスタック **FaceStack** を用意し、そこに多角形面素を重なり順に格納することを行う。addFace 関数では自身を再帰的に呼び出すことで、順番に面をスタックに追加するが、面の重なり順として妥当でない面が追加された場合はスタックからポップし、次の面を格納することを行う。これにより、全ての重なり順を調べるよりは遙かに効率よく妥当でないものを判定の対象からはずすことができる。

```
addFace() {
```

```
foreach(FaceStack に含まれない Face f) {  
  // f を FaceStack の末尾に追加してよいか判定  
  if (FaceStack.canAddFace(f)) {  
    FaceStack.push(f);  
    if (全ての面の順番が決まったら) {  
      処理を終了; // 妥当な解が見つかった  
    }  
    addFace(); // 再帰的に次の層へ  
  }  
}  
if (FaceStack.empty()) {  
  処理終了; // 妥当な解が見つからなかった  
} else {  
  FaceStack.pop();  
}
```

図4 多角形面素の重なり順の推定アルゴリズム

全ての多角形面素を問題なくスタックに格納し終わったら、妥当な重なり順が求まったものと判断できる。解が1つも見つからなかった場合は、展開図が局所平坦条件の(4)を満たしていないことがわかる。図4の addFace 関数では、スタックに面を追加してよいか否かを判定するための関数 canAddFace を呼び出しているが、この関数はスタックの最上位に、引数の多角形面素 F を追加できる場合に true を、そうでない場合に false を返すものとする。ここで、スタックに F を追加できる場合とは、 F を追加しても次の2つの条件が満たされる場合である。

- ・ 折線での折り曲げ方が展開図の山折、谷折と矛盾しない
- ・ 任意の断面で面の交差が生じない

上記の条件の判定方法を説明する前に、面の稜線を CE:Contour Edge (輪郭線), UE:Upper Connect Edge (上に接続する稜線) および LE:Lower Connect Edge (下に接続する稜線) に分類する。面の表側が上を向いている場合、谷折の稜線は LE, 山折の稜線は UE となる。面が下を向いている場合はこの逆となる(図5)。面の表側が上下のどちらを向いているかは前章の折りたたみ操作で確定している。

折紙の多角形面素の重なり順に、スタックに下から順番に面を追加していくことを考えると、追加する面 F に LE を介して隣接する面 F' が、必ずスタック内に存在する必要がある(スタック内に F' が存在しない場合、 F よりも F' が上方に配置されることになり、LE の谷折、山折が展開図と矛盾することとな

る). つまり, このことが上記(1)の条件である.

(2)の条件の判定では, 新しく配置する面 F が既にスタックに格納されている面 F' の UE の接続を妨害しなければよい.

各面 F' に含まれる未接続の UE (以降では e と表現する) を, 面 F が覆い隠す位置にある場合(図 6(a))は false となる. (図 6 では, 最上位の面を太線で表している. また灰色の領域は理解しやすくするために幅を持たせてあるが幾何的には幅ゼロの領域である.)

e が面 F の UE と同一の位置にある場合(図 6(b))は true である.

e が面 F の LE と同一の位置にある場合, 面 F と接続する面 F_{neighbor} との位置関係によって条件が異なる.

- F_{neighbor} より下の場合(図 6(c))は true である.
- F_{neighbor} より上の場合で, F_{neighbor} と F' が重なり合う場合(図 6(d))は false である.
- F' が F_{neighbor} より上の場合で, F' と F_{neighbor} が重なり合わない場合(図 6(e))は true である.

図 5 多角形面素の重なり順の制約(点線は谷折)

図 6 多角形面素の重なり順の制約(点線は谷折)

5. 3. 算出手法の実装結果

以下に, スタッキングオーダーの推定を実験した結果を示す. CPU: Pentium Mobile Processor2.0GHz, RAM: 1.0GB の PC に実装したシステムで実験を行った. 対象とした展開図は図 7 に示す 4 つで, それぞれ(a)は単純な例, (b)は中割折りの例, (c)は兜, (d)は鶴の半分の展開図である. 鶴に関しては展開図が対称な形状であることを考慮して, その半分の展開図に対して評価を行った. 実験結果は表 1 に示す通りである. 表中の N は多角形面素の数, N' は面の重なりが妥当であるかの

判定を行った回数, #Ans は得られた解の数, time は計算に要した時間(秒単位)である. 図 7 中の多角形面素に振られている数字は, 最初に見つかった解の面の重なり順を表している.

図 7 重なり順の推定結果(複数解のうちの 1 つ)

表 1 計算結果

	N	$N!$	N'	#Ans	time(s)
(a)	4	24	10	1	0.06
(b)	8	4.0×10^4	79	3	0.09
(c)	18	6.4×10^{15}	8.3×10^4	2778	0.76
(d)	26	4.0×10^{26}	3.4×10^6	5.0×10^5	89

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るが, 他に 2 通りの折り方があることが判明し, 興味深い結果を得ることができた(図 7(b)の解は, 中割りではない折り方の例である). 兜の例では, 4 万近くの解が抽出された. これは, 互いに関与しない面の重なりが存在する場合, 解となる重なり順が増大するためである. 鶴の例では, 多角形面素の数が 26 であり, 全ての場合は実時間で計算しきれないが, 効率的に不要な判定を除くことができた結果, 判定の数は 3.4×10^6 に収まっている. しかし, この例でも兜の例と同じ理由で解の数が膨大になっている. これらの解の中から, 折りたたんだ結果の形状が同一のものを除去し, 本質的に異なるものを抽出する後処理を実装する必要がある.

6. ユニークな解の抽出

互いに重なりあわない面どうしは、スタック内での上下関係は意味を成さない。従って、互いに重なりあう位置にある面間の上下関係のみに着目し、それらが同一であれば、折紙の作品としては同一な解であると言えることができる。

前章で得られた複数の解からユニークな解を数上げるために、以下に示すような行列の定義を行う。

まず、折りたたんだ後の2次元空間において、互いに重なっているか否かを表す、次のような $n \times n$ 行列 M_{mask} を作成する。 n は折紙に含まれる構成面素の数である。 M_{mask} の各要素の値 m_{ij} は次のように定める。

$$m_{ij} = \begin{cases} 1 & i \neq j \text{ and } F_i \cap F_j \neq \phi \\ 0 & \text{otherwise} \end{cases}$$

式中の $F_i \cap F_j \neq \phi$ は面 F_i と面 F_j が互いに重なっていることを意味する。

次に、得られたスタッキングオーダーに基づいて、各面同士の上下関係を行列 M_{order} で表現する。 M_{order} の要素の値 m_{ij} は次のように定める。

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

$\text{Order}(F)$ は面 F のスタッキングオーダーであり、 $\text{Order}(F_i) > \text{Order}(F_j)$ であれば、面 F_i が面 F_j より上にあることを意味する。

続いて、次のように成分同士のかけ算を行う演算子 $\otimes$ を定義する。

$$A \otimes B = C$$

のとき、

$$c_{ij} = a_{ij} b_{ij}$$

以上の定義に基づき、 $M_{\text{mask}} \otimes M_{\text{order}}$ を行くと、互いに重なる位置にある面素同士間の上下関係だけを表した行列が得られる。従って、前章で得られる各解を表す M_{order} に対して $M_{\text{mask}} \otimes M_{\text{order}}$ を求め、この結果のユニークな解の数を数え上げればよい。

6. 1. ユニークな解の抽出の具体例

図に示す例を元に、どのような行列を生成し、ユニーク解を得るか以下に説明する。

図 8 重なり順の推定結果(複数解のうちの 1 つ)

まず、折りたたみ形状を推定後に、互いの構成面素の位置から、重なっている面素の関係を調べる。図の例では、 F_1 と重なっているものは、 F_2, F_3, F_4 であり、 F_2 と重なっているものは F_1, F_4 、 F_3 と重なっているものは F_1 、 F_4 と重なっているものは F_1, F_2 である。従って、 M_{mask} は次のようになる。

$$M_{\text{mask}} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

ところで、4 章で述べた総当たりの方法で得られるスタッキングオーダーの解は以下の 3 通りである。

解[0]: $1 < 3 < 2 < 4$

解[1]: $1 < 2 < 3 < 4$

解[2]: $1 < 2 < 4 < 3$

これらの解のそれぞれの行列 M_{order} は次のように表される。

$$M_{\text{order}}^{[0]} = \begin{pmatrix} 0 & -1 & -1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & -1 & 0 & -1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad M_{\text{order}}^{[1]} = \begin{pmatrix} 0 & -1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ 1 & 1 & 0 & -1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

$$M_{\text{order}}^{[2]} = \begin{pmatrix} 0 & -1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & -1 & 0 \end{pmatrix}$$

それぞれの行列に対し、 M_{mask} との演算を施すと、以下のようにすべてが同じものになる。

$$\mathbf{M}_0 \otimes \mathbf{M}_{mask} = \mathbf{M}_1 \otimes \mathbf{M}_{mask} = \mathbf{M}_2 \otimes \mathbf{M}_{mask}$$

$$= \begin{pmatrix} 0 & -1 & -1 & -1 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

従って、ユニーク解は1つであることがわかる。

6. 2. ユニークな解の抽出の実装結果

本章で提案した手法を実装した結果を示す。実行においては、 $\mathbf{M}_{mask}$ は必ず対称行列になり、また $\mathbf{M}$ は右上の要素(m_{ij} , $i < j$)がわかれば左下の要素(m_{ji} , $i > j$)が一意に決まるため、計算時には右上要素だけを考慮すれば良い。

図7に示す例題に対して行った、ユニークな解を算出した結果を表2に示す。ユニークな解は#Ans'である。この値は実際に紙を折って正しい結果であることを確認した。兜には9通りの場合が存在すること、鶴の半分の展開図では1通りしかユニーク階が存在しないことがわかった(鶴の全体の展開図ではこの限りではない)。

表2 計算結果

	N	N'	#Ans	#Ans'	time(s)
(a)	4	10	1	1	0.22
(b)	8	79	3	3	0.31
(c)	18	8.3×10^4	2778	9	1.0
(d)	26	3.4×10^6	5.0×10^5	1	110

7. まとめと展望

展開図を折りたたんだ後の構成面素の配置(2次元上の位置)を決定する問題、実現可能なスタッキングオーダーを算出する問題、得られたスタッキングオーダーからユニークな解を数え上げる問題のそれぞれについて、解法を提示し、それを計算機上の実装した結果を示した。

互いに重なり合わない面の組み合わせが多いと、ユニークな解は少ないにもかかわらず、妥当なスタッキングオーダーが膨大になってしまい、無駄な計算が大部分を占めることとなる。この問題を解決するために、局所的なスタッキングオーダーの情報を有効に活用すれば、明らかに結果が重複するスタッキングオーダーの数え上げを減らすことができ、より複雑な展開図に対してもユニーク解を算出できるようになると考えられる。

参考文献

- [1] Bern, M. and Hayes, B.: The complexity of flat origami, Proceedings of the seventh annual ACM/SIAM symposium on Discrete algorithms (1996), pp. 175-183.
- [2] 川崎敏和: バラと折り紙と数学と, 森北出版株式会社(1998).
- [3] 深川英俊: 折紙の数学, 森北出版株式会社 (2002).
- [4] Lang, R. J.: Origami Design Secrets: Mathematical Methods for an Ancient Art, AK Peters, Ltd. (2003).
- [5] 内田忠, 伊藤英則: 折り紙過程の知識表現とその処理プログラムの作成, 情報処理学会論文誌, Vol. 32, No. 12 (1991), pp. 1566-1573.
- [6] T. Hull. On the mathematics of flat origamis. Congressus Numerantium 100 (1994), 215--224.
- [7] 島貫博, 加藤ジェーン, 渡邊豊英: 展開図を用いた折り紙操作過程における手順毎の折り方の構成, 電子情報通信学会技術研究報告, Vol. 102, No. 55 (2002), pp. 71-78.
- [8] Shimanuki, H., Kato, J. and Watanabe, T.: Construction of 3-D Paper-made Objects from Crease Patterns, in Proc. of IAPR Conference on Machine Vision Applications (MVA2005), pp. 35-38 (2005).
- [9] 三谷純: 折紙展開図エディタ ORIPA. <http://mitani.cs.tsukuba.ac.jp/pukiwiki-origa>
- [10] Kawasaki, T.: On the Relation Between Mountaincreases and Valley-creases of a Flat Origami, Proceedings of the First International Meeting of Origami Science and Technology (1989), pp. 229-237.
- [11] Justin, J.: Towards a Mathematical Theory of Origami, Proceedings of the Second International Meeting of Origami Science and Scientific Origami (1997), pp. 15-29.
- [12] Hull, T.: On the Mathematics of Flat Origamis, Congressus Numerantium, Vol. 100 (1994), pp. 215-224.

著者紹介

三谷 純 : 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻(兼任所属機関 科学技術振興機構 さきがけ), 305-8573, 茨城県つくば市天王台 1-1-1