Origami Folding Sequence Generation Using Discrete Particle Swarm Optimization

Ha-Duong Bui[1(✉)], Sungmoon Jeong[1], Nak Young Chong[1,2], and Matthew Mason[2]

[1] School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa, Japan 923-1211
{bhduong,jeongsm,nakyoung}@jaist.ac.jp
[2] School of Computer Science and Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA 15213
matt.mason@cs.cmu.edu

Abstract. This paper proposes a novel approach to automating origami or paper folding. The folding problem is formulated as a combinatorial optimization problem to automatically find feasible folding sequences toward the desired shape from a generic crease pattern, minimizing the dissimilarity between the current and desired origami shapes. Specifically, we present a discrete particle swarm optimization algorithm, which can take advantage of the classical particle swarm optimization algorithm in a discrete folding action space. Through extensive numerical experiments, we have shown that the proposed approach can generate an optimum origami folding sequence by iteratively minimizing the Hausdorff distance, a dissimilarity metric between two geometric shapes. Moreover, an in-house origami simulator is newly developed to visualize the sequence of origami folding.

Keywords: Mathematical Origami, Folding Sequence Generation, Combinatorial Optimization Problem, Particle Swarm Optimization

1 Introduction

Many innovative products, such as bendable electronics, deployable solar array, foldable paper lithium-ion battery, are inspired by origami, the art of paper folding. From the 1930s, problems related to the folding and unfolding have attracted a large attention in the computational geometry community [10]. Recent advances in computing environment allow researchers to study complex folding systems that include programmable matters [8, 11] and folding machines [3, 7]. However, few researches have addressed the following issues: (1) how to find a crease pattern of a desired origami shape without going through the unfolding process, (2) how to generate a folding sequence to create a desired shape from a generic crease pattern.

Three main areas of origami in computational geometry, related to folding problems are origami simulator, folding sequence generation, and multiple object

folding from a single sheet, respectively. *Freeform Origami* [14–16] is a well-known folding simulator providing users with an environment to interact with a virtual paper. Users are able to fold papers and interactively modify crease patterns, and generate crease patterns for polyhedra. Using the quadrilateral mesh information of the 3D input shape, *Freeform Origami* first unfolds the input shape to get the candidate crease pattern which is then simultaneously folded and controlled through affine transformations. This process is iterated until the desired shape is achieved. Another interesting research performed by Akitaya et al. [1] is how to generate folding sequences of flat-foldable origami. To accomplish this goal, the framework builds a new graph-like data structure called the extended crease pattern. This data is constructed using the input crease pattern by which the input shape is unfolded. The folding sequence can then be found by inverting the unfolding process. However, users are often requested to decide the next step in the unfolding sequence, because multiple outcomes are possible from the extended crease pattern. Therefore, it is considered a semi-autonomous system. An et al. [2] studied how the 3D shapes can be transformed using the programmable matter. They designed a programmable sheet with a set of hinges. Multiple shapes can be constructed and transformed using the programmable sheet, without considering any folding action sequences. In the above-mentioned literatures, origami problems and their solution approaches are mainly studied from a mathematical perspective. It is important to note that crease patterns were folded into desired shapes, but the crease patterns could only be obtained through the unfolding process of the desired shapes.

In this paper, we aim to present an algorithmic framework for automating the folding process. We show how to find the folding sequence from a generic crease pattern. A generic crease pattern could possibly folded into multiple shapes. Our approach tries to find an optimal solution through the sequential combination of appropriate folding actions. Specifically, in our problem definition, the inputs are a generic crease pattern and a desired shape, and the output is a folding sequence. Our proposed algorithm can create folded shapes from the given generic crease pattern, maximizing the similarity with the desired shape. We formulate our problem as a combinatorial optimization problem (**COP**) and employ a discrete particle swarm optimization (**DPSO**) algorithm. The proposed approach will be described in detail in the following sections.

2 Methodology

2.1 Problem Formulation

Problem Definition Figure 1 shows a diagram of the proposed folding automation system. The input to our system consists of

1. A square sheet of paper, $OriginalPaper$
2. A set of n predefined creases, $ActionSet = \{Action_1, Action_2, ..., Action_n\}$
3. A desired folded shape, $InputObject$

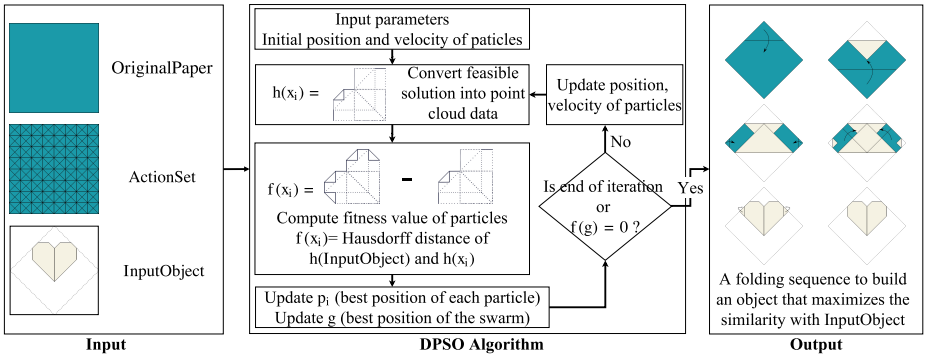


Fig. 1. Overview of the proposed approach

All of the following are outputs from the proposed system.

1. An optimal folding sequence g from the given ***generic*** crease pattern
2. A final folded shape $OutputObject$ achieved by applying the folding sequence g to $OriginalPaper$
3. The similarity score between $InputObject$ and $OutputObject$

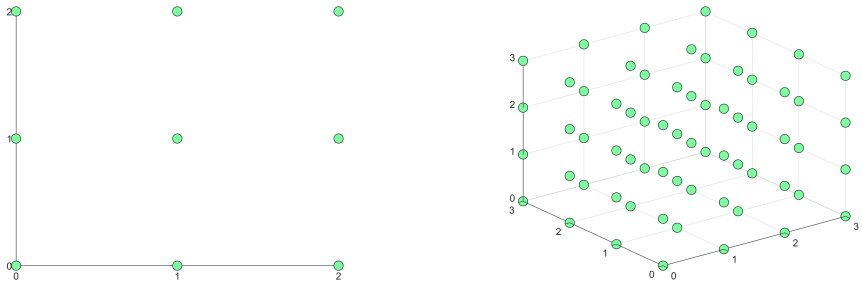
Feasible Solution We define a nominal folding action called $Action_0$ causing the current shape to remain unchanged. Adding the $Action_0$ to the $ActionSet$, our new $ActionSet$ becomes

$$ActionSet = \{Action_0, Action_1, Action_2, ..., Action_n\}$$

A folding process is a list of actions chosen out of $ActionSet$, meaning that the list of actions are sequentially applied to the original paper to be folded into an object shape.
From the above description, a **feasible solution** x is defined as a folding process with a fixed length n given by

$$\begin{cases} x = (x_1, x_2, ..., x_n) \\ x_i \in ActionSet, 1 \leq i \leq n \end{cases}$$

Search Space We define a search space A which is a set of all feasible solutions. Typically, A is a $n - dimensional$ integer lattice, denoted by $\mathbb{Z}^n$, which is the lattice in the Euclidean space $\mathbb{R}^n$. Because each candidate solution x is a vector of n elements, and we also have $n+1$ candidates for each element, then the set A has a cardinality (the total number of feasible solutions) of $(n+1)^n$. The size of the search space A increases exponentially with the number of folding actions. Figure 2 illustrates the search space A and the feasible solutions with $n = 2$ and $n = 3$, respectively.



(a) $n = 2$ (b) $n = 3$

Fig. 2. Examples of search space. (a) With $n = 2$, 9 feasible solutions exist. (b) With $n = 3$, 64 feasible solutions exist. The green dots are feasible solutions.

Objective Function Let $h(x)$ be a function that converts a feasible solution x into an object shape (Sect. 2.3). Also let $f(x) : \mathbb{Z}^n \rightarrow \mathbb{R}$ be a function that assesses the degree of similarity between $h(x)$ and $InputObject$. A lower value of $f(x)$ indicates a high degree of similarity with $InputObject$, and vice versa. The function f is the objective function to be minimized. We can now formulate our problem as an optimization problem in the following way:

Given $f : A \rightarrow \mathbb{R}$ *from* A *to the real number,*
Seek an element g *in* A *such that* $f(g) \leq f(x)$ *for all* x *in* A.

2.2 Problem Optimization: DPSO Algorithm

We choose DPSO as a meta-heuristics algorithm that can provide an approximate solution to our problem. A basic variant of the PSO [9] algorithm contains a list of feasible solutions called particles which are members of a swarm. These particles are initially given their positions and velocities randomly. Each particle employs the objective function to evaluate its position. It also uses its velocity to move to new positions. Here the moving function should be carefully designed by users. Moreover, (1) the particle's current direction, (2) the particle's best-known position, as well as (3) the swarm's best-known position, are combined to update the particle's velocity. When a particle discovers a new best position, it will communicate and update the entire swarm. The process of updating and relocation of the swarm is repeated, and by doing so, it is expected, but not guaranteed, that an optimal solution will eventually be explored.

As having described, obviously, the most important things in PSO algorithm are the particle's velocity and position. The behavior of particles is directly affected by the velocity. In order to move, three information are processed that are (1) particle's current direction, (2) particle's previous best position and (3) swarm's best-known position [4]. In this research, we redefined these concepts

and mathematical operations. This discretization method helps adjust the classical PSO's properties and make it capable of searching in the discrete search space of our problem.

The above description can be formalized by the following moving functions,

$$\begin{cases} v_i^{t+1} \leftarrow \omega \otimes v_i^t \oplus \varphi_p r_p \otimes (p_i^t \ominus x_i^t) \oplus \varphi_g r_g \otimes (g^t \ominus x_i^t) \\ x_i^{t+1} \leftarrow x_i^t \oplus v_i^{t+1}, \end{cases} \tag{1}$$

where

- v_i^t velocity at time step t of particle i
- x_i^t position at time step t of particle i
- p_i^t particle i's best known position at time step t
- g^t swarm's best known position at time step t
- $\omega, \varphi_p r_p, \varphi_g r_g \sim U(0,1)$ social/cognitive confidence coefficients
- $a \otimes b = a \times b \mod (n+1)$
- $a \oplus b = a + b \mod (n+1)$
- $a \ominus b = a - b \mod (n+1)$

The pseudo code of DPSO is introduced in Algorithm 1.

Algorithm 1 Discrete Particle Swarm Optimization

```
for all particle i do
    x_i ← InitializeParticle(n)
    for all dimension d do
        v_{i,d} ~ U{0, n}
while termination condition not reached do
    for all particle i do
        Pick random numbers r_p, r_g ~ U(0,1)
        Update particle's velocity and position using Eq. (1)
        Evaluate the fitness f(x_i)
        if f(x_i) < f(p_i) then
            p_i ← x_i
            if f(p_i) < f(g) then
                g ← p_i
g is the proposed solution of the algorithm
```

2.3 Converting Folding Sequences into Object Shapes

In order to convert a feasible solution x into an object shape (function $h(x)$ in Sect. 2.1) and visualize the proposed sequence of folding, we develop an in-house origami simulator. This system was originally presented by Miyazaki et al. [12]. We can construct flat folding origami shapes from generic crease patterns.

Specifically, we use faces, edges, and vertices to describe the state of the folded paper. Each flat sheet of paper is represented by a face. A face contains multiple edges. Moreover, each edge has two vertices. A folded paper is a list of faces.

Based on the idea that each crease will separate the origami plane into two half-planes, we construct a binary tree structure, in which the root is the $OriginalPaper$. Each node of the tree is a face and stores the information about the relative position of the plane with the original plane. When applying a folding action, we traverse from the root and find all the leaves that contain the crease, make these leaves become parent nodes, and create two new children nodes (as two new faces which are created by the folding action). Obviously, two nodes with the same parent will share one common edge. Finally, we combine all the leaves of the binary tree to get the final shape.

An example is shown in Fig. 3, where we apply the folding actions in the following order: $Action_1 \rightarrow Action_2 \rightarrow Action_3$. E1, E2, E3, E4, and E5 are the common edges between faces. When we combine all the leaves F5, F6, F7, F8, F9, and F10, we can get the object shape. The states of the origami shape after each folding action are shown in Table 1.

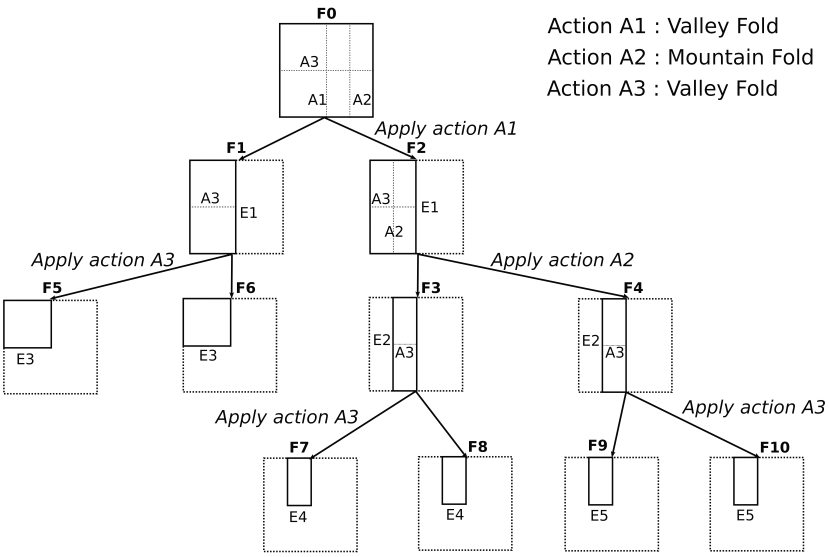


Fig. 3. Example of converting a feasible solution into an object shape

We also implement a function to calculate the similarity among folded origami papers using the point cloud data (function $f(x)$ in Sect. 2.1). With the PCL library [13], we convert the object shapes into the point cloud data based on the information about faces, edges, and vertices of shapes. We normalize the point cloud data in size and density. Then, the Hausdorff distance [6] between two

Table 1. The states of the origami shape after each folding action in Fig. 3

Applied action	Leaf nodes	Parent nodes	Common edges
Begin	$\{F_0\}$	$\emptyset$	
$Action_1$	$\{F_1, F_2\}$	$\{F_0\}$	$\{F_1 \cap F_2 = E_1,$
$Action_2$	$\{F_1, F_3, F_4\}$	$\{F_0, F_2\}$	$F_3 \cap F_4 = E_2, F_5 \cap F_6 = E_3,$
$Action_3$	$\{F_5, F_6, F_7, F_8, F_9, F_{10}\}$	$\{F_0, F_1, F_2, F_3, F_4\}$	$F_7 \cap F_8 = E_4, F_9 \cap F_{10} = E_5\}$

object shapes can be calculated. The lower the Hausdorff distance, the higher the similarity between two shapes, and vice versa.

3 Experimental Evaluation

3.1 Experimental Set-up

Original Origami Paper The original paper used in our experiments is a flat square paper with a size of 60 × 60 *units*. The front side of the paper is blue and the back side is white.

Predefined Crease Patterns We use four predefined crease patterns in our experiments as shown in Fig. 4. Each crease represents a valley fold or a mountain fold.

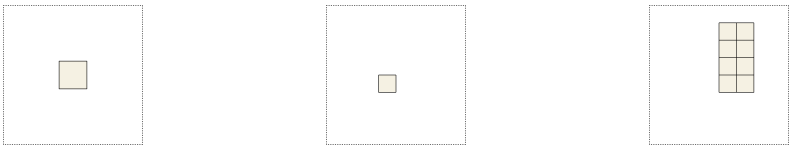
(a) *Orthogonal* CP_1 (b) *Orthogonal* CP_2 (c) *Orthogonal* CP_3 (d) *Interlaced CP*

Fig. 4. Crease patterns used in experiments

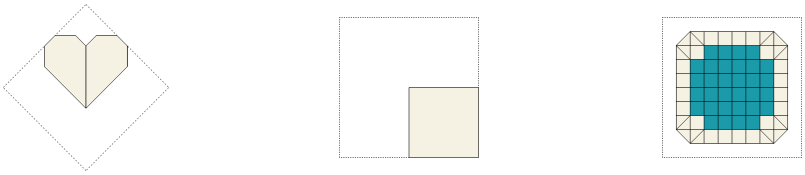
Experimental Objective Models Using the same folding simulation was introduced in Sect. 2.3, we fold the desired shapes shown in Fig. 5. In the figures, the black dashed lines represent the original paper.

DPSO Parameters Referring to *Standard Particle Swarm Optimization* [5], we choose the parameters of DPSO algorithm as follows:

- The swarm size $S = 35 + B(10, 0.5)$. $B(10, 0.5)$ is a binomial distribution, where 10 is the number of trials and 0.5 is success probability in each trial.
- The maximum number of iterations is 100



(a) Desired Shape 1 (b) Desired Shape 2 (c) Desired Shape 3



(d) Desired Shape 4 (e) Desired Shape 5 (f) Desired Shape 6

Fig. 5. Desired shapes used in experiments

- $\omega \simeq 0.721$
- $\varphi_p = 1$
- $\varphi_g = 1$

3.2 Experimental Results

We report the experimental results obtained from the 6 test sets in Table 2.

Table 2. Results of experiments

	Predefined crease pattern	Objective model	Haursdoff distance			Iteration			Avg. runtime (ms)
			Min	Max	Avg	Min	Max	Avg	
E1	*Orthogonal* CP_1	Desired Shape 1	0.0	0.0	0.0	1	4	1.06	838.24
E2	*Orthogonal* CP_2	Desired Shape 2	0.0	0.0	0.0	1	90	18.33	35064.85
E3	*Orthogonal* CP_2	Desired Shape 3	0.0	3.75	0.07	1	100	20.50	33893.63
E4	*Interlaced CP*	Desired Shape 4	0.0	5.32	2.98	1	100	73.42	45533.28
E5	*Orthogonal* CP_1	Desired Shape 5	8.5	8.50	8.50	100	100	100	78594.77
E6	*Orthogonal* CP_2	Desired Shape 6	3.3	16.37	6.0	100	100	100	164159.70

In the experiments E1-E4, the folding sequence of the desire shape is a subset of *ActionSet*, which means we can fold the desire shape from the given predefined crease pattern. It was confirm that if the desired shape is built using a set of creases chosen out of the generic input crease pattern, then our method can always find the optimal solution.

Furthermore, in the experiments E1-E4, different types of shapes (square, rectangular and heart shape) are employed to evaluate our system. As expected,

our experiments prove that with a suitable generic crease pattern, we can create arbitrary shapes. Moreover, our system also provides an efficient way to transform shapes.

To evaluate the cases where the desired shapes are not built from the folding actions of *ActionSet*, we perform the experiments E5 and E6. In the experiment E5, the desired shape in Fig. 5e is a square with a size of $\frac{1}{4}OriginalPaper$. Obviously, we cannot construct any shape that is the same as the desired shape from the *Orthogonal* CP_1. In Fig. 6a, the proposed solution of our algorithm for the experiment E5 is shown. Because attempting to get 0.0 in Hausdorff distance is impossible, the shapes that are most similar to the desired shape are presented. Correspondingly, in the experiment E6, the desired shape is neither a rectangle nor a square but an octagon. We need to find a folding sequence to build it from the *Orthogonal* CP_7. The shape of the optimal solution has been obtained and introduced in Fig. 6b. To summarize, our approach is quite successful in finding the minimum Hausdorff distance between the desired shape and the feasible solutions.

(a) Output Shape of E5 (b) Output Shape of E6

Fig. 6. Output shapes of experiments E5 and E6

From the test cases, we see that the computational time increases with the number of creases. There are two reasons for it. First, the calculation time of the folding simulation is dependent on the size of the *ActionSet*. The more folding actions we apply to the origami paper, the longer time the program needs to build the object shape. Secondly, as we discussed in Sect. 2.1, the search space exponentially grows with the input size. Therefore, the computational costs for arriving at feasible solutions in the search space A increase accordingly as the input size increases.

4 Conclusion

We presented a combinatorial optimization approach to automating origami. The DPSO algorithm was employed to solve the folding problem. Moreover, an in-house origami simulator was newly developed to perform extensive numerical

experiments. Our method was verified to show an efficient technique to generate feasible folding sequences in an autonomous way. We have confirmed that, with a properly prepared generic crease pattern, various types of origami shapes can be folded easily with the proposed method.

References

1. Akitaya, H.A., Mitani, J., Kanamori, Y., Fukui, Y.: Generating folding sequences from crease patterns of flat-foldable origami. In: ACM SIGGRAPH 2013 Posters. pp. 20:1–20:1. SIGGRAPH '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2503385.2503407
2. An, B., Benbernou, N., Demaine, E.D., Rus, D.: Planning to fold multiple objects from a single self-folding sheet. Robotica 29(1), 87–102 (Jan 2011)
3. Balkcom, D.J., Mason, M.T.: Robotic origami folding. The International Journal of Robotics Research 27(5), 613–627 (2008), http://dx.doi.org/10.1177/0278364908090235
4. Clerc, M.: Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem, pp. 219–239. Springer Berlin Heidelberg, Berlin, Heidelberg (2004), https://doi.org/10.1007/978-3-540-39930-8_8
5. Clerc, M.: Standard Particle Swarm Optimisation (Sep 2012), https://hal.archives-ouvertes.fr/hal-00764996, 15 pages
6. Deza, M.M., Deza, E.: Encyclopedia of distances. In: Encyclopedia of Distances, pp. 1–583. Springer (2009)
7. Felton, S., Tolley, M., Demaine, E., Rus, D., Wood, R.: A method for building self-folding machines. Science 345(6197), 644–646 (2014), http://science.sciencemag.org/content/345/6197/644
8. Hawkes, E., An, B., Benbernou, N.M., Tanaka, H., Kim, S., Demaine, E.D., Rus, D., Wood, R.J.: Programmable matter by folding. Proceedings of the National Academy of Sciences 107(28), 12441–12445 (2010), http://www.pnas.org/content/107/28/12441.abstract
9. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Neural Networks, 1995. Proceedings., IEEE International Conference on. vol. 4, pp. 1942–1948 vol.4 (Nov 1995)
10. Margalit, F.: Akira Yoshizawa, 94, Modern Origami Master. The New York Times (Apr 2005)
11. Miyashita, S., Guitron, S., Ludersdorfer, M., Sung, C.R., Rus, D.: An untethered miniature origami robot that self-folds, walks, swims, and degrades. In: 2015 IEEE International Conference on Robotics and Automation (ICRA). pp. 1490–1496 (May 2015)
12. Miyazaki, S., Yasuda, T., Yokoi, S., Toriwaki, J.i.: An origami playing simulator in the virtual space. The Journal of Visualization and Computer Animation 7(1), 25–42 (1996), http://dx.doi.org/10.1002/(SICI)1099-1778(199601)7:1<25::AID-VIS134>3.0.CO;2-V
13. Rusu, R.B., Cousins, S.: 3d is here: Point cloud library (pcl). In: 2011 IEEE International Conference on Robotics and Automation. pp. 1–4 (May 2011)
14. Tachi, T.: Freeform Rigid-Foldable Structure using Bidirectionally Flat-Foldable Planar Quadrilateral Mesh, pp. 87–102. Springer Vienna, Vienna (2010), https://doi.org/10.1007/978-3-7091-0309-8_6
15. Tachi, T.: Freeform variations of origami. J. Geom. Graph 14(2), 203–215 (2010)
16. Tachi, T.: Freeform origami. www.tsg.ne.jp/TT/software/ (2010–2016)