A NEW DESIGN METHOD FRAMEWORK FOR OPEN ORIGAMI DESIGN PROBLEMS

A Dissertation

by

WEI LI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2014

Major Subject: Mechanical Engineering

ABSTRACT

With the development of computer science and manufacturing techniques, modern origami is no longer just used for making artistic shapes as its traditional counterpart was many centuries ago. Instead, the outstanding lightweight and high flexibility of origami structures has expanded their engineering application in aerospace, medical devices, and architecture. In order to support the automatic design of more complex modern origami structures, several computational origami design methods have been established. However these methods still focus on the problem of determining a crease pattern to fold into an exact pre-determined shape. And these methods apply deductive logic and function for only one type of topological origami structure.

In order to drop the topological constraints on the shapes, this dissertation introduces the research on the development and implementation of the abductive evolutionary design methods to open origami design problems, which is asking for their designs to achieve geometric and functional requirements instead of an exact shape. This type of open origami design problem has no formal computational solutions yet.

Since the open origami design problem requires searching for solutions among arbitrary candidates without fixing to a certain topological formation, it is NP-complete in computational complexity. Therefore, this research selects the genetic algorithm (GA) and one of its variations – the computational evolutionary embryogeny (CEE) – to solve origami problems.

The dissertation made two major contributions. One contribution is on creating the GA-based/abstract design method framework on open origami design problems. The other contribution is on the geometric representation of origami designs that directs the definition and mapping of their genetic representation and physical representation. This research introduced two novel geometric representations, which are the “ice-cracking” and the pixelated multicellular representation (PMR). The proposed design methods and the adapted evolutionary operators have been testified by two open origami design problems of making flat-foldable shapes with desired profile area and rigid-foldable 3D water containers with desired volume. The results have proved the proposed methods widely applicable and highly effective in solving the open origami design problems.

ACKNOWLEDGEMENTS

At first, I would like to thank my advisor and my committee chair, Dr. McAdams, for his instruction and support during these three years, and for providing me the opportunity to work on this research about origami. I am also grateful to my committee, Dr. Amato, Dr. Malak, and Dr. Rathinam, for their guidance and suggestions on this research.

Thanks also go to my friends and colleagues – Shraddha Sangelkar, Michael Glier, Jonathan Arendt, Madison Burns, Sooyeon Lee, Joanna Tsenn, Elissa Morris, Wesley Jones, Aeriel Corey, and William Tanner Parten – making my time at Texas A&M University a fantastic experience.

I also want to extend my gratitude to the National Education Foundation, which provided financial support for this research project.

Finally, I thank my parents for their encouragement and understanding on both my study and my life in the United States.

NOMENCLATURE

CEE	Computational Evolutionary Embryogeny
CEEFOOD	CEE for Optimal Origami Design
CRA	Crease Restoration Algorithm
GA	Genetic algorithm
MV-assignment	Mountain Valley Assignment
PMR	Pixelated Multicellular Representation

TABLE OF CONTENTS

LIST OF FIGURES

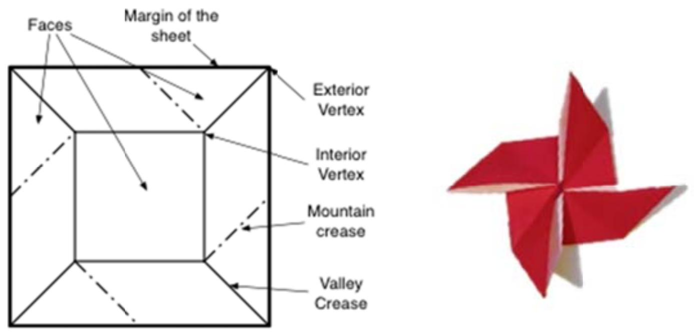
LIST OF TABLES

CHAPTER I

INTRODUCTION

Popular since the 17th century, origami design is traditionally a handcrafted art form (Harbin 1997). Ever since the 1700s, many origami artists have created and published their hand-made origami works (Fuse 1990, Harbin 1997, Hull 2006, Lang 2003, Montroll 1979). More recently, researchers have discovered origami's connections with, and applications in, engineering (Gantes 2001). The field of origami engineering is growing with the development of formal mathematical and computational methods for designing folded shapes. Additionally, origami-style geometries have found multiple applications in engineered systems. Within the last three decades, origami engineering has advanced into several branches including origami art design (Lang 2003), origami mathematics (Demaine and Demaine 2001), and computational origami (Fastag 2006, Tachi 2006). Nevertheless, the two fundamental origami problems - origami design and foldability (Demaine and Demaine 2001) – remain only partially solved. Folding pattern design refers to the design of a crease pattern for a flat sheet. If a flat sheet is folded along the specified creases a specific shape is formed. In the foldability problem, the crease pattern has been given. The question that remains for foldability is - whether there exists a valid mountain-valley assignment for this pattern to achieve the intended shape.

In geometry, origami structures are made up of a series of folds in a sheet of paper (Figure 1). The locations of the folds are formally called "*creases*". The various locations, folding directions and folding sequence of the creases determine the ultimate shape of the structure. Creases can be described by their lengths, or by their endpoints – generally *“vertices”*. In this dissertation, any vertex that falls on the margins of the sheet is called an *“exterior vertex”*; otherwise, the vertex is called an *“interior vertex”*. We also consider the spaces between the creases, known as *“faces”*. To determine the direction of a fold along a crease, a mountain-valley assignment (MV-assignment) is typically used. With a *mountain* fold, the crease is at the top, and the paper will be folded to the back of itself. But with a *valley* fold, the crease is at the bottom, and the paper is folded to the front.



(a) (b)

Figure 1 Terminologies for origami crease pattern. (a) A pinwheel base crease pattern; (b) The folded pinwheel.

This dissertation will focus on origami design automation. On the study of origami design automation, several computational methods (Demaine, Demaine and Ku 2011, Demaine and O'Rourke 2007, Hull 2006, Lang 1996, Lang 2003, Tachi 2006, Tachi 2010) have been developed by other researchers. Those methods, such as TreeMethod (Lang 1996), Origamizer (Tachi 2006, Tachi 2010), and etc., generally set up the laws for mapping the desired folds in the origami shapes back to the local crease pattern molecules, as well as the means for connecting these molecules to form the whole crease pattern. However, most of the above computational methods are focusing on deriving crease patterns for given final folded origami shapes, but leaving the sculpturing of the folded shapes to the origami artists or engineers.

To synthesize the folded shape design into computational origami design and optimization process, this dissertation will present evolutionary methods that design origami structures including geometric, functional, and foldability properties. Without a clear definition of the required folded origami shape, the formation of vertices and creases is not definitive either. Thus the origami topological structures with undetermined crease patterns would cause the design search space to have high non-linearity, unpredictable dimensionality, and non-convexity. Moreover, the origami design problems that synthesize the folded shape design often involve multiple objectives. For multi-objective design and optimization problems, most of the conventional solution search methods have limitations in dealing with non-convex design space, disconnected Pareto frontier, or non-differential objective functions. Evolutionary algorithms, in contrast, have proved to be less susceptible to the shape or continuity of the Pareto frontiers, and more effective with little a priori understanding of the search space (Coello 2009, Kicinger, Arciszewski and De Jong 2005). As a result, the genetic algorithm, which is a widely applied evolutionary algorithm on system design and optimization, will be used for this research. This dissertation will adapt genetic algorithms through solving two key problems prior to applying GA to the design problems. One problem is on how to use binary genetic codes to encode any candidate solution, while the other is on how to accelerate and regulate the evolution process.

Studies on origami folding often assume that the shape can be folded flat (Mitani 2011). Flat-foldability is not only an important property that makes industrial origami structures more compact and portable, but also fits most of the traditional origami artworks that are made only by folding and unfolding (Mitani 2011). To be a flat-foldable origami, the crease folds must be locally flat-foldable everywhere, which means each vertex and its connected creases needs to satisfy the Maekawa-Justin theorem, the Kawasaki-Justin theorem, and the local min theorem (Demaine and O'Rourke 2007). Also, as a whole, the origami structure cannot collide with itself during the folding procedure. Since this dissertation is on new applications of GA and GA-based meta-heuristics, I will restrict my study to flat-foldable origami structures, on which we have more comprehensive understanding. This dissertation will also present a heuristic for flat-foldability identification of a known crease arrangement. The basic idea of the heuristic is to tree-search all the MV-assignments and face overlapping orders, while immediately pruning the “branches” of solutions as they violate the local flat-foldability (Theorem 1(1)) or the non-collision criteria (Theorem 1(3)). Since the flat-foldability problem is proved NP-hard, the above tree-search heuristic has to go through a “brute force” process with clear guidance and simplification.

In summary, the research topics that will be covered in this dissertation are given in Figure 2. The main topics are the two major research fields on origami design automation and self-folding origami control, the three specific applications, as well as the GA-based methods. Several subordinate research topics that support the main topics are also listed in the figure. The subordinate topics of the GA-based methods include the individual encoding methods, the fitness ranking approaches, and the evolutionary operators design, which are boxed by dotted lines. The octagons mark three subsidiary research topics – origami flat-foldability, data clustering, and linear classification – that will be introduced and adapted as the mathematical tools to support the main research topics.

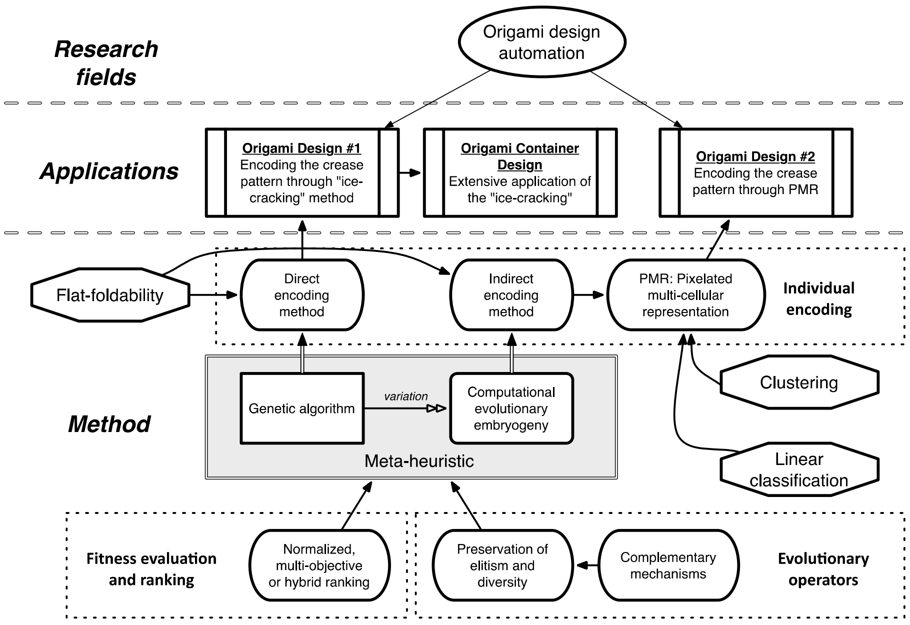

Research fields
Origami design automation
Applications
Origami Design #1
Encoding the crease pattern through "ice-cracking" method
Origami Container Design
Extensive application of the "ice-cracking"
Origami Design #2
Encoding the crease pattern through PMR
Flat-foldability
Direct encoding method
Indirect encoding method
PMR: Pixelated multi-cellular representation
Individual encoding
Clustering
Method
Genetic algorithm
variation
Computational evolutionary embryogeny
Meta-heuristic
Linear classification
Fitness evaluation and ranking
Normalized, multi-objective or hybrid ranking
Preservation of elitism and diversity
Complementary mechanisms
Evolutionary operators

Figure 2 The research fields, applications, and the GA and CEE methods that are studied in this dissertation

Design Problem Statement

Origami engineers and artists have successfully found viable computational methods (Demaine, Demaine and Ku 2011, Lang 1996, Lang 2005, Tachi 2010) to develop ways of deriving crease patterns for desired folded origami shapes, but few of these existing methods actually design the folded shape of an origami structure. An example origami design problem that considers the finished folded shape design is given below.

Objective:

Design a flat-foldable origami structure that has a minimal change on the center of mass between its spread state and folded state. We assume that at the spread state, the origami sheet lies in a horizontal plane P_h; while at the folded state, all the faces stand in a vertical plane P_v. During the folding procedure, one of the creases must be anchored steady, so that the two planes P_h and P_v will intersect at this anchored crease.

Constraints:

1. The spread state origami sheet has a given shape S_t;
2. The number of creases falls within a certain range (M_L, M_U);
3. The flat-folded state profile has a certain area A_t.

The constraints S_t, M_L, M_U, and A_t are four to-be-determined arguments (TBD arguments) for specific problems. The spread state shape S_t defines the shape of the unfolded sheet that will be folded. For traditional origami works, S_t is usually a square, a 1:2 rectangle, or an isosceles right triangle. In the research cases of cube folding (Amato, Dill and Song 2003, Amato and Song 2002) and shopping bag folding (Balkcom, Demaine and Demaine 2004), $\mathrm{S_t}$ will need to define some irregular but continuous 3D surfaces. The area of flat-folded state profile $\mathrm{A_t}$ is a unique property possessed only by flat-foldable origami. For instance, the shopping bags need to be able to fold flat for the convenience on mass storage and transportation. Therefore, the flat-folded profile of a shopping bag must be smaller and less distorted. The range of the number of creases $(\mathrm{M_L}, \mathrm{M_U})$ is not as directly related with the physical performance of the origami structures as $\mathrm{S_t}$ and $\mathrm{A_t}$, but it is a factor that defines the complexity of the origami crease pattern (another factor is how the creases are linked). More creases in the crease pattern result in more complexity, while fewer creases mean less complexity. Less complexity in the origami crease pattern with as few creases as possible is intuitively preferred for either artistic or engineering applications, since origami structures with fewer creases will be much easier to manufacture and fold. However, a crease pattern with too few creases will be less able to satisfy some of the design objectives. For instance, it is impossible to fold a container for holding water using a crease pattern with only two creases. In practice, the lower limit $\mathrm{M_L}$ is always set to 1, unless the designer knows the minimum number of creases that is enough for realizing the design requirements. The upper limit $\mathrm{M_U}$ is determined according to the limitation of manufacturing or manual crafting skill. The function of $\mathrm{M_L}$ and $\mathrm{M_U}$ is to confine the design search space. As a rule of thumb, narrowing down the range of $(\mathrm{M_L}, \mathrm{M_U})$ speeds the optimization method.

Toward the objective and constraints of the design problem, I use the following objective function:

$$
\begin{aligned}
H(\widehat{\boldsymbol{V}}, l, \widehat{\boldsymbol{C}}, m, \widehat{\boldsymbol{F}}, n) &= \sum \overline{H_i}(\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_p) \times w_i + P(\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_p) \\
&= \overline{H_e}(\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_p) \times w_e + \overline{H_a}(\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_p) \times w_a + \\
&\quad \overline{H_f}(\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_p) \times w_f + P(\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_p) \\
&= \widehat{H_e}(\widehat{\boldsymbol{V}}, l, \widehat{\boldsymbol{C}}, m, \widehat{\boldsymbol{F}}, n) \times w_e + \widehat{H_a}(\bar{A} - A_t) \times w_a + \\
&\quad \widehat{H_f}(Err_{flat-foldability}) \times w_f + \hat{P}(m)
\end{aligned}
$$

$$
\boldsymbol{Soft\ constraints}: \begin{cases} Crease\ pattern\ is\ flat - foldable \\ m < M_U \\ S(\widehat{\boldsymbol{V}}, l, \widehat{\boldsymbol{C}}, m, \widehat{\boldsymbol{F}}, n) = \bar{S}(\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_p) = \boldsymbol{S}_t \\ A(\widehat{\boldsymbol{V}}, l, \widehat{\boldsymbol{C}}, m, \widehat{\boldsymbol{F}}, n) = \bar{A}(\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_p) = A_t \end{cases}
$$

where $\overline{H_e}$ is the evaluation of the potential energy (which equivalently measures the location of center of mass) at the flat-folded state, $\overline{H_a}$ represents how different the folded state profile area and the target value of $5/18$ are, $\overline{H_f}$ represents the evaluation of global flat-foldability error (as defined in (Li and McAdams 2013)), P is the penalty term due to the disagreement with soft constraints, S is the spread state profile, and A is the flat-folded state profile area. The overall fitness value H is formulated as a weighted sum of its components of $\overline{H_e}$, $\overline{H_a}$, and $\overline{H_f}$ that are related to the properties of the candidate solution and the soft constraints as well. If the number of creases exceeds the restricted range defined by M_L and M_U, an extra "penalty" term $P(\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_p)$ will also be added.

In addition, since the origami design problem won't constrain the exact shape of the origami, this research will define the type of such problems as ***open origami design problems***.

Computational Complexity

The computational complexity is a measurement for the inherent difficulty of a problem or an algorithm (approach). The computational complexity of a problem is the minimum of the computational complexity of all of its possible approaches.

However, some of the algorithm will convert or simplify its problem. TreeMaker converts the desired shape design problem to the design of a uniaxial base. Origamizer simplifies the shape surface design problem to the realization of a triangular mesh. The computational complexity of these methods depends on their converted problems. The design of a uniaxial base has been proven to be equivalent to a disk packing problem (Lang 2003). Thus the complexity of TreeMaker is that of the corresponding disk packing algorithm that it applies. The realization of a triangular mesh is equivalent to the rearrangement of the triangular facets in a plane without overlapping and the insertion of tuck molecules. So the complexity of Origamizer is that of seeking facet arrangement and tuck molecule assignment.

For a design problem as described in the prior section, the crease pattern designs are arbitrary. So the solution search space is theoretically infinite. For simplification, we only allow the vertices to be chosen from the nodes of a square lattice with a resolution of r-by- r. There will be about r^2 available positions for each vertex to locate. Under the simplification, if the search for the crease pattern design has to try every planar graph (Trudeau 1994), in which any vertex within the square origami paper has at least four linked creases (edges). Without using faster algorithms, this solution-search procedure can further be decomposed to three stages:

Stage 1: Determine the number of vertices ($\mathrm{n_V}$) and the number of creases ($\mathrm{n_C}$).

Stage 2: Place the vertices ($\mathrm{V} = \{\mathrm{v_i}, \mathrm{i} = 1,2, \ldots, \mathrm{n_V}\}$) on the available locations within and on the border of the origami paper.

Stage 3: Connect the vertices by the creases ($\mathrm{C} = \{\mathrm{c_i}, \mathrm{i} = 1,2, \ldots, \mathrm{n_C}\}$) that have no intersections with each other.

When we apply restrictions on $\mathrm{n_V}$, $\mathrm{n_C}$, as well as on the available locations for placing the vertices, stages 1 and 2 are combinatorial search problems. Upon the completion of stages 1 and 2, stage 3 can be converted to a basic NP-complete problem (Garey and Johnson 1979). Here I use $\mathrm{E} = \left\{\mathrm{e_j} = \left\{\mathrm{v_{j1}}, \mathrm{v_{j2}}\right\} \subseteq \mathrm{V}\right\}$ to represent the union of all the edges, whose two end points are from the vertices V. Then the crease must be a chosen as a subset of the edges, which means $\mathrm{C} \subseteq \mathrm{E}$. In this case, I build a dual graph $\overline{\mathrm{G}} = \left(\overline{\mathrm{V}}, \overline{\mathrm{E}}\right)$, where $\overline{\mathrm{V}} = \mathrm{E}$ and $\overline{\mathrm{E}} = \mathrm{R} = \{\forall \mathrm{r_k} = \{\mathrm{e_{k1}}, \mathrm{e_{k2}}\} \subseteq \mathrm{E}, \mathrm{e_{k1}}\ \text{doesn't intersect with}\ \mathrm{e_{k2}}\}$. As a result, stage 3 is no simpler than the NP-complete problem of looking for a clique of size $\mathrm{n_C}$ or more within the dual graph $\overline{\mathrm{G}}$ (Garey and Johnson 1979).

Through the problem decomposition and analysis, I conclude that the crease pattern design problem is an NP-complete problem. In order to solve such a problem, a heuristic is necessary to avoid any form of exhaustive search and accelerate the solution derivation process.

Dissertation Formation

In the following chapters of this dissertation, I will illustrate my research methodology on solving the NP-complete origami design problem. The heuristic that I select is an evolutionary algorithm. My research topic is to implement evolutionary algorithms to design a generic origami design method, which solves problems that request for a broad range of geometric and functional requirements.

In chapter II, I will go through some established research that is either related to my research topic or essential for inspiring my research methodology. This research includes widely studied origami design methods, and the study on variations of evolutionary algorithms.

In this chapter I have defined the objective function of the design problem. In chapters III and IV, I will introduce my solutions for the other two key aspects of applying evolutionary algorithms – the geometric representation of candidate designs and the evolutionary operators.

In chapter V, I present designs derived by the evolutionary algorithms. And I will make a discussion through the comparison of the designs and through analyzing the whole solution derivation procedure. The discussion will reveal the advantages and limitations of the evolutionary algorithms that I will propose for the design problem.

In chapter VI, I implement and adapt the same evolutionary algorithm for another origami design problem. The new problem asks for the design of an origami water container that has a desired volume. The results of the evolutionary algorithm will be used to shown the extensiveness and the applicability of the method for variations of open origami design problem.

CHAPTER II

LITERATURE REVIEW

The earliest record of traditional origami can be dated back to the 17th century. But modern origami study has only a few decades' history. Modern origami designers emphasize designing models, which have not only good final shapes but also good sequences. The two most significant symbols that mark modern origami are wet-folding and the standard Yoshizawa-Randlett system (Wikipedia 2014). Wet-folding was first introduced by Akira Yoshizawa - a prolific origami grandmaster, who was inspired by modern renaissance art. Wet-folding makes the manipulation of paper easier and results in origami works with a more sculpted look. The Yoshizawa-Randlett system is a diagramming system used to describe the folds of origami models. Such description of basic origami techniques for constructing origami models is widely used in present origami textbooks.

Then since the 1960s, a larger amount of artists and engineers have started to pay extra attention to origami and the remarkable structural advantages of origami mechanisms. As a general term for origami research and study, origami engineering includes five major foci – mathematics in origami, origami design automation, computational simulation & visualization, origami structures in engineering, and origami in education. Based on the design problem description given in the last chapter, this dissertation will mostly present my work on two core topics about mathematics in origami and origami design automation. The computational simulation & visualization is somewhat relevant, but I will use it as a tool to support the study on the two core research topics.

In this chapter, I will mainly go through the established research and results of origami engineering that are closely relative with my dissertation study. The content of this chapter will be partitioned into five sections: origami foldability, origami design methods, intelligent origami, evolutionary design, and genetic algorithm.

In the section of origami foldability, I will introduce the mathematical and geometric theorem and results that will be applied in my research. And the introduction mainly covers the results on flat-foldability and rigid-foldability of origami structures, as these two features are requested in my research problem.

The next section will list and introduce some of the most studied origami design methods. I will also provide a summary of these methods and reveal why they cannot be adapted to my research problem.

The third section on intelligent origami is an extension of the fundamental research. I will introduce the studies that make the origami folding or designing automated by innovative approaches. These studies have inspired some of my ideas in realizing origami design automation.

The fourth and the fifth sections are about the research methodology that I use for solving my design problem. Since the design problem has been proven NP-complete, and we are seeking for a generic approach, the evolutionary design applying an abductive measure for generating designs is one suitable choice. The two sections will present the studies on the basic evolutionary algorithms, as well as some new founding on the applications of novel evolutionary algorithm variations.

Origami Foldability in Mathematics and Geometry

Origami design can be decomposed into three major sub-problems, which are crease pattern design, MV-assignment, and final shape design Figure 3. Crease pattern design involves the crease arrangement design, as well as the mountain-valley-assignment (MV-assignment) of all the creases. The creases arrangement design tells where in the origami sheet the creases (folds) will be placed, while the MV-assignment determines each crease to be either a mountain or a valley that constrains the direction of crease folding. The folding angle design is an extension of the MV-assignment design. The folding angles of all the creases are required for uniquely defining the folded state of an origami structure as long as the folding is not flat. The folding sequence is literally the sequence of making the folds. For many of the cases, some creases have to be folded at the same time; but simple folds usually can be folded separately. The final shape design needs to derive the proper folding angles and folding sequence of the creases to transform the origami sheet into the desired shape. Although not a part of origami design, the design of a geometric representation is also an essential component, since it defines the language and the syntax that are used by designers and computers to describe an origami structure.

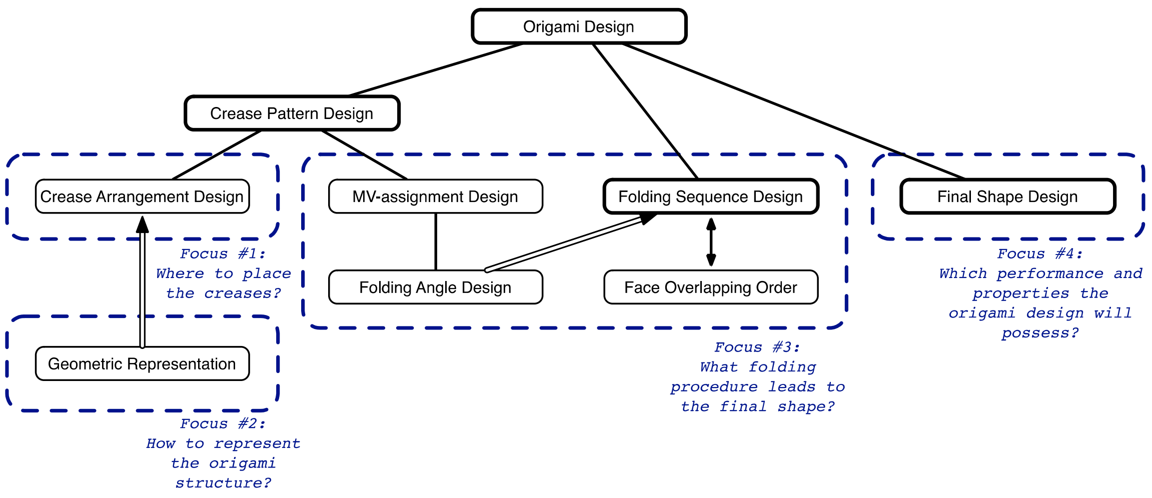

Origami Design
Crease Pattern Design
Crease Arrangement Design
MV-assignment Design
Folding Sequence Design
Final Shape Design
Folding Angle Design
Face Overlapping Order
Geometric Representation
Focus #1: Where to place the creases?
Focus #2: How to represent the origami structure?
Focus #3: What folding procedure leads to the final shape?
Focus #4: Which performance and properties the origami design will possess?

Figure 3 The sub-problems under origami design.

Origami foldability is an element of origami engineering different than general origami design. And it is closely related with and required by all the sub-problems of origami design. Though the foldability serves as one of the performance criteria for general origami design (in most cases). However, foldability can be studied independently based on some existing origami mathematics results. Based on the mathematics and geometry in origami, the most studied and used foldability requirements are flat-foldability and rigid-foldability.

Flat-foldability

For the flat-foldability of origami structures, several origami researchers have created similar theorems to identify it. According to Demaine, O’Rourke (Demaine and O'Rourke 2007), Poma (Poma 2009), and Schneider (Schneider 2004), flat-foldability can be defined by local and global scopes. The local flat-foldability, which considers only the small region around each vertex, has already been mathematically defined as the union of three basic origami theorems, which are the Maekawa-Justin theorem, the Kawasaki-Justin theorem, and the local min theorem (Demaine and O'Rourke 2007). Global flat-foldability implies the local flat-foldability on every vertex and a non-collision condition. Justin (Esquivel, Xing, Collier, Tomaso et al. 2011) and Konjevod (Konjevod 2006) listed three possible situations that breaks the non-collision condition. Poma (Poma 2009) and Schneider (Schneider 2004) have also presented their mathematical interpretations of non-collision condition in their manuscripts. However, the inspection of the global flat-foldability, or say the inspection of the global non-collision condition, is known to be NP-hard (Bern and Hayes 1996), no matter what method has been used. Besides the studies given above, Hull (Hull 1994, Hull 2006) has also studied flat-foldability. He has proved the edge-2-colorability of flat-foldable crease patterns. As a summarization, (Li and McAdams 2013) reformulates the check of three face surface crossings (Esquivel, Xing, Collier, Tomaso et al. 2011) to the identification of two penetration conditions, and provides a practicable tree-search heuristic to check the global flat-foldability for crease patterns with known MV-assignments.

According to the research of Demaine, O’Rourke (Demaine and O'Rourke 2007), Poma (Poma 2009), and Schneider (Schneider 2004), globally flat-foldability must satisfy the following theorem.

Global flat-foldability theorem: An origami structure is globally flat-foldable, if and only if its crease pattern satisfies:

(1) (Necessary condition) Locally flat-foldable everywhere;

(2) (Lemma of condition 1) 2-colorable; and

(3) (Sufficient condition) Capable of stacking all faces flat without causing collision or penetration between faces and creases.

In Theorem 1, condition (1) is the requirement for the creases. It says the creases that intersects on any vertex, needs to satisfy the Maekawa-Justin theorem, the Kawasaki-Justin theorem, and the local min theorem for origami (Demaine and O'Rourke 2007). The third Theorems say:

Maekawa-Justin theorem: If one looks inside a flat origami without unfolding it, one sees a zigzagged profile, determined by an alternation of "mountain creases" and "valley creases". The numbers of mountains and valleys always differ by 2, which means $|\mathrm{m_M} - \mathrm{m_V}| = 2$, where $\mathrm{m_M}(> 0)$ is the number of mountain creases, and $\mathrm{m_V}(> 0)$ is the number of valley creases.

Kawasaki-Justin theorem: A given crease pattern can be folded to a flat origami if all the sequences of angles $\{\mathrm{a_1}, \mathrm{a_2}, \ldots, \mathrm{a_{2n}}\}$ surrounding each (interior) vertex fulfill the following condition

$$a_1 + a_3 + \cdots + a_{n-1} = a_2 + a_4 + \cdots + a_n = \pi$$

or

$$a_1 - a_2 + a_3 - a_4 + \cdots + a_{n-1} - a_n = 0$$

Local Min theorem: In any flat folding, any wedge whose angle is a local min (smaller than the angles of its two adjacent wedges) must be delimited by one mountain and one valley fold.

Condition (2) is for the faces. In a flat-foldable origami structure, using only two colors must be enough for dyeing the faces, so that no adjacent faces will share a same color. This condition stands a lemma of the three theorems in condition 1. And it is also a necessary condition for global flat-foldability.

The condition (3) declares the geometric relation of faces and creases during the dynamic folding process. A mathematical expression for condition (3) is specifically explained in Schneider’s manuscript (Schneider 2004).

Throughout the origami design procedure, the flat-foldability identification needs to start with a single given crease arrangement. To complete the crease pattern, all MV-assignments that are locally flat-foldable must be derived first. Then, for each MV-assignment, we also need to provide its folding sequence to reveal its global flat-foldability. For origami structures with simple folds only, the order for creases to be folded one by one is enough to define the folding sequence; otherwise multiple creases may need to fold simultaneously. In the latter situation, the order of face overlapping at the flat folded state is sufficient for defining how the origami structure is folded than a folding sequence. Any occurrence of face penetration (collision) indicates that the crease pattern is not foldable by using the current face overlapping order.

In a crease arrangement, only the location and angle of each crease are given. An MV-assignment is required to further define which directions these creases will be folded to with respect to its adjacent faces. Therefore, to determine whether an MV-assignment is invalid for flat-foldability, we check if it violates Theorem 1(1). I apply an exhaustive tree search algorithm to get all the locally flat-foldable MV-assignments from the crease arrangement. With a given arrangement of m creases, the exhaustive tree search algorithm for deriving all locally flat-foldability MV-assignments theoretically has a $O(2^{\mathrm{m}})$ step bound. However, when we practically perform the exhaustive tree search, any newly grown leaf that will later develop into a sub-tree of candidate MV-assignments will be immediately pruned, as long as the leaf has violations on Theorem 1 condition (a).

Face overlapping orders will then be based on locally flat-foldable MV-assignments. For any given crease arrangement, either there is more than one locally flat-foldable MV-assignment (at least two with one being a reverse of the other), or no locally flat-foldable MV-Assignments are possible. For each MV-assignment there are many possible and distinct folding sequences, so faces in flat-folded state could thus be stacked in a different order as well. Here, I again use a similar exhaustive tree search algorithm to determine face overlapping orders. In accordance with the third condition of Theorem 1, we exhaustively search the existence of face penetration for each possible face overlapping order. The definition of face penetration and the algorithm of checking penetration are explained as follows.

The penetration check algorithm requires us to firstly calculate the semifolding of an origami structure $\mu\colon \mathrm{R}^2 \rightarrow \mathrm{R}^2$, which is adapted from Schneider's method (Schneider 2004). The semifolding procedure is physically processed through the following 3 steps.

1) Have the crease pattern ready on a sheet of paper, whose thickness is small enough to be ignored during any folding procedure, and place it in an horizontal R^2 plane;

2) Cut the paper along all the creases to derive the separated origami faces;

3) Rearrange and stack the faces parallel in the same horizontal R^2 plane, where the uncut sheet was in, then flip some of them if necessary, until all the conditions below satisfy:

    a) If two faces are adjacent in the crease pattern, the edges that used to be their common crease must coincide;

    b) For every face that hasn't been flipped, its adjacent face, which was linked by a valley crease, must have been flipped and stacked above;

    c) For every face that hasn't been flipped, its adjacent face, which was linked by a mountain crease, must have been flipped and stacked below.

Following the semifolding procedure, I give the equivalent mathematical expressions for the semifolded origami structure. As a convention for our notation, a bar is used to distinguish between the features such as vertices, creases and faces and the semifolding remaps of those features. For instance, the i-th crease $\mathrm{C_i}$ to $\overline{\mathrm{C}}_\mathrm{i}$(i.e. $\overline{\mathrm{C}}_\mathrm{i} = \mu(\mathrm{C_i})$). For convenience of explanation, at the semifolded state, we call the two adjacent faces of any crease $\overline{\mathrm{C}}_\mathrm{i}$ as the "cover faces" of the crease $\overline{\mathrm{C}}_\mathrm{i}$. Since we have a definite order of face overlapping, we further name the cover face that stacks above another to be the upper cover face $\overline{\mathrm{F}}_{\mathrm{i,cover,U}}$. The one below is consequently named the lower cover face $\overline{\mathrm{F}}_{\mathrm{i,cover,L}}$. The faces stacking between two cover faces are called the content faces $\overline{\mathrm{F}}_{\mathrm{i,content}}$ of crease $\overline{\mathrm{C}}_\mathrm{i}$. All the content faces are thus above the lower cover face, but below the upper cover face.

The face penetration check algorithm implements an exhaustive search sequence, which goes through every crease. For the crease $\overline{\mathrm{C}}_{\mathrm{i}}$, its two adjacent faces will be the cover faces. The content faces can also be determined.

At the semifolded state of the crease pattern, I simplify the three collision cases in (Schneider 2004) down to the following two situations of face penetration. The first situation is when a crease $\overline{\mathrm{C}}_{\mathrm{j,k}}$ of one content face $\overline{\mathrm{F}}_{\mathrm{i,content,j}}$ intersects with the crease $\overline{\mathrm{C}}_{\mathrm{i}}$;

$$\overline{\boldsymbol{C}}_{j,k} \cap \overline{\boldsymbol{C}}_{i} \neq \emptyset, \overline{\boldsymbol{C}}_{j,k} \text{ } is \text{ } a \text{ } crease \text{ } of \text{ } \overline{\boldsymbol{F}}_{i,content,j}. \tag{1}$$

The second situation is when the crease $\overline{\boldsymbol{C}}_{i}$is totally within the region of a content face $\overline{\boldsymbol{F}}_{i,content,j}$, but has no intersection with any of the creases of that content face;

$$\overline{\boldsymbol{C}}_{i} \cap \overline{\boldsymbol{F}}_{i,content,j} = \overline{\boldsymbol{C}}_{i} \text{ } and \text{ } \overline{\boldsymbol{C}}_{common} \cap \boldsymbol{\Omega}\overline{\boldsymbol{F}}_{i,content,j} = \emptyset \tag{2}$$

or

$$\overline{\boldsymbol{C}}_{j,k} \cap \overline{\boldsymbol{F}}_{i,content,j} = \overline{\boldsymbol{C}}_{i} \text{ } and \text{ } \overline{\boldsymbol{C}}_{i} \cap \overline{\boldsymbol{C}}_{j,k} = \emptyset, \forall \overline{\boldsymbol{C}}_{j,k} crease \text{ } of \text{ } \overline{\boldsymbol{F}}_{i,content,j}. \tag{3}$$

Every possible MV-assignment and face overlapping order for a given crease pattern is tested with the penetration check algorithm. As soon as face penetration is found, the penetration check stops and labels the tested combination of MV-assignment and face overlapping order as not flat-foldable. Again, the exhaustive tree search for face overlapping order based on one crease pattern has a theoretical $\mathrm{O}\big((\mathrm{n}-1)!\big)$ step bound, where n is the number of faces. But the occurrence of face penetration will cause the search tree to be pruned early thus in practice narrowing the search space.

If no face penetration is spotted after a thorough search that has considered all the creases as the common crease for the penetration check, the crease pattern with the given combination of MV-assignment and face overlapping order is found to be flat-foldable. Finally, if a crease pattern has no flat-foldable combination of MV-assignment and face overlapping order, it is defined as not flat-foldable.

More details about the approaches for detecting the flat-foldability of an origami design will be described in following chapters.

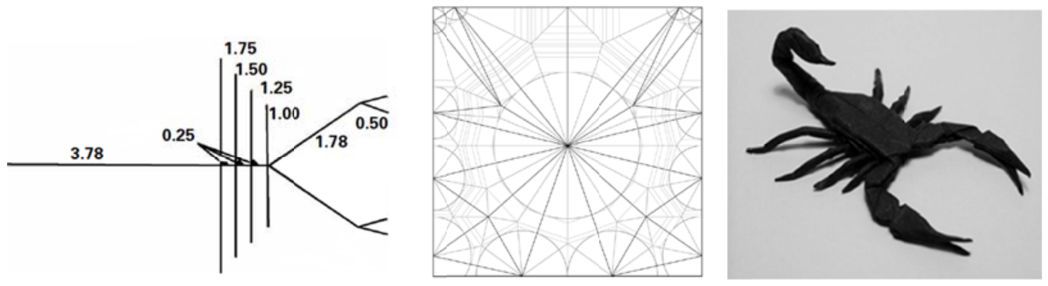
Rigid-foldability

Rigid-foldability requires each face of an origami shape to stay non-deformed throughout the entire folding procedure. The rigid-foldability has been studied by D. A. Huffman (Huffman 1976), T. Hull (Hull 2006), T. Tachi (Tachi 2006, Tachi 2010), etc. The fundamental methodology for inspecting rigid-foldability is to simulate the entire folding procedure. For arbitrary crease pattern design, the folding of the shape is unpredictable. Thus it is also impossible to tell when and where there will be face collision without actually calculating the folded shape and searching for possible conflicts among the non-penetrable faces at every moment from the initial state to the final state. The simulation of folding will partition the entire procedure into intervals. And after each time interval, I re-calculate all of the folding angles and check face collision. In chapter VI, I will provide the algorithm for inspecting rigid-foldability in detail.

Origami Design Methods

Several effective computational methods have been developed to achieve crease pattern designs for origami structures. Though these methods don’t explicitly consider the folding sequence design, their resultant crease patterns always implies a valid folding sequence. In this section, I will introduce the three most well-formed origami design methods, and summarize their common limitation when encountering an open design problem as the one given in the first chapter.

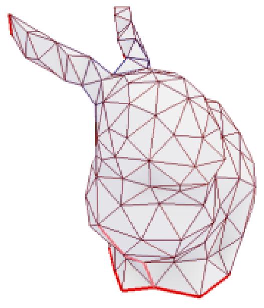
The TreeMaker algorithm (Lang 1996, Lang 2005) requires the designer to first simplify his desired origami shape into an approximate tree graph. If at all possible, this method then automatically generates a crease pattern that can be folded into a shape with a projection of the tree graph. Though TreeMaker usually generates a “fiendishly” complex crease pattern that is difficult to fold by hand, it is a great tool for any origami artist or engineer to get a rough prototype for a design. To make the compositions look more artistic, designers have to do additional manual operations, like squeezing, bending, or tucking, on the final shapes. Figure 4 shows the products at the end of the three steps of TreeMaker . The design procedure starts with a desired subject to realize. In this case, the subject is a scorpion. Then the designer has to manually make a hypothetical tree graph Figure 4(a), where each branch roughly corresponds to one limb of the finished design. The tree graph will be sent to the TreeMaker software to derive a crease pattern like Figure 4(b). Since the arrangement of nodes is equivalent to the disk packing problem, the algorithm is able to produce different crease patterns instead of the global optimal one. The designer has to interact with the computer to derive a favorable crease pattern design. After that, the last step is to print the crease pattern out, and fold it. Directly folding the crease pattern, we can only obtain a base with uniaxial flaps. So in order to “volumize” the shape, we need to – again - manually squeeze or crumple the flaps to form a shape like Figure 4(c). If it is impossible or difficult to fold the uniaxial base into a desired shape, the designer has to go back to the tree graph or the TreeMaker software to make some modifications. In all, TreeMaker is a powerful tool that is capable of generating artistic designs, but it also requires lots of experience of manual origami design from the designers.

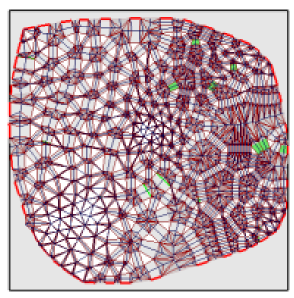


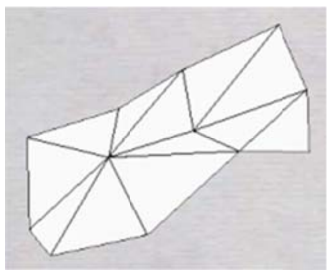
(a) (b) (c)

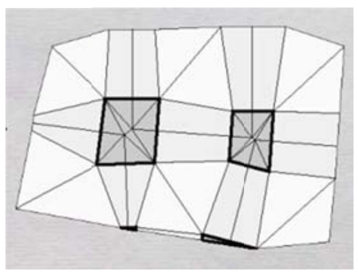
Figure 4 The procedure of TreeMaker for deriving an origami shape design from a hypothetical tree graph. (a) To design an origami scorpion, the TreeMaker starts with a manual design of a hypothetical tree graph, where each branch will roughly correspond to one limb in the final shape; (b) A crease pattern will be derived by TreeMaker algorithm for the tree graph; (c) Fold the crease pattern and make modification, an origami scorpion is finished.

Tachi, Demaine, et al. presented a method to "origamize" 3-D polyhedral Figure 5. Unlike TreeMaker, which builds the body of an origami shape, "Origamizer" uses the sheet to form the surfaces of a shape (Tachi 2006). The method requires a model's surface mesh being divided into flat triangular facets at first (as described in (Tachi 2006)). Then it tries to arrange these facets into a planar sheet, by adding jointed tucking molecules that connect them. This method has good efficiency and usually succeeds after several failed trials. The surface division is thus the most important factor that determines the success of “Origamizer”.


(a)


(b)


(c)


(d)

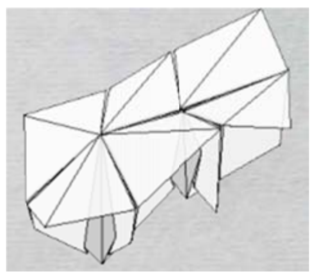
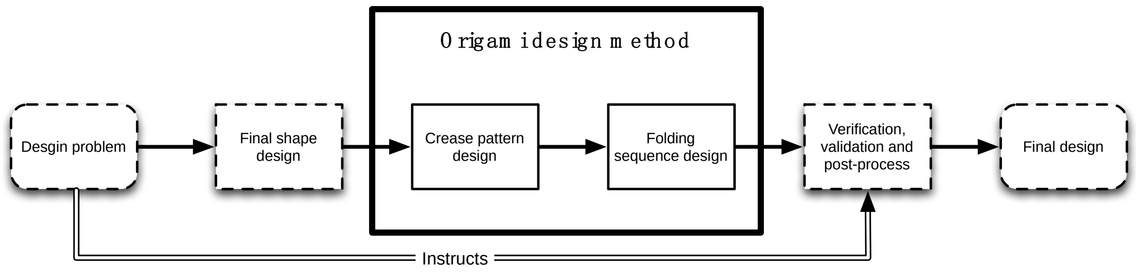

(e)

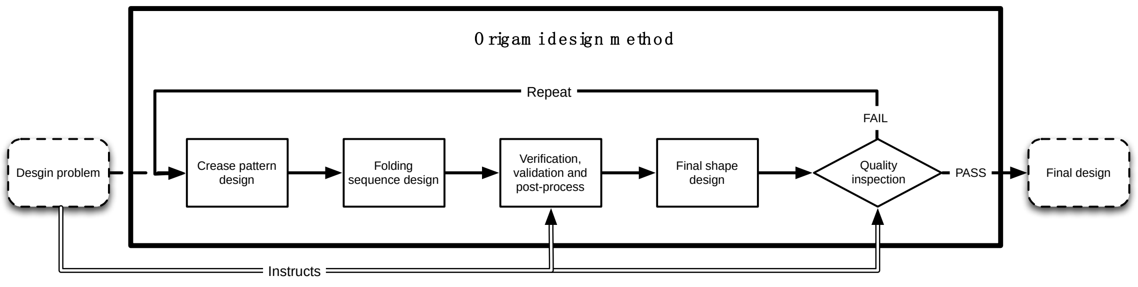
Figure 5 Origamizer. (a) The surface of a rabbit shape has been divided into triangles; (b) Origamizer derives a crease pattern for the triangular meshed surface of the rabbit; (c) A surface with 12 triangular facets; (d) The triangular facets are flattened and rearranged in a plane, and tuck molecules are inserted to connect them and form a foldable crease pattern; (e) The folded shape of the crease pattern in (d). The tuck molecules are folded underneath the triangular facets to the ribs.

Demaine et al. has developed methods for making orthogonal structures, such as mazes (Demaine, Demaine and Ku 2011) and “cube gadgets” (Ovadya 2010), from simple universal hinge patterns. Their method divides the whole flat sheet into square grids. Then they design a set of crease pattern molecules, so that each square block in the grid can be assigned one of the molecules. A proper assignment scheme will result in a full crease pattern for the desired orthogonal structure. Although the design is constrained to a highly modularized origami consisting of orthogonal origami gadgets with fixed sizes, the method is very efficient in deriving the crease pattern.

There are three common features of the three computational origami design methods listed above. One feature is that they require the design objective to satisfy some specific geometric constraints. TreeMaker can only design shapes based on uniaxial origami shapes; Origamizer is constrained to realize meshed surfaces; and the orthogonal maze/cube folding is only for shapes made of orthogonal surfaces. Another feature is that these methods need the final folded shape of the origami design to be given in a problem. Therefore, if the final folded shape is not explicitly defined in a design problem, the above methods are not applicable. Finally, the third common feature is that these design methods are deductive. They directly build crease patterns based on the geometric analysis of desired shapes. The workflow of a deductive origami design method is shown as Figure 6(a). Starting with the design problem, a deductive method generally requires that the final shape design can be hypothetically framed out based on the geometric requirements. According to the geometric formation, a proper design method or algorithm will be selected to directly derive the crease pattern as well as the folding sequence that lead to the final design. In summary, deductive design methods are more like delicately designed mapping algorithm between particular origami shapes and their crease patterns. The advantage of deductive methods is the high accuracy and high efficiency in achieving desired shapes. Thus they are great tools for artists to assist the manual origami design and sculpturing. The limitation of the deductive methods is also quite obvious that they lack a generic solution for arbitrary origami designs.



(a)



(b)

Figure 6 The workflow diagrams of deductive origami design methods (a) and abductive origami design methods (b).

The abductive origami design methods, on the other hand, will make logical guesses and improve the guesses progressively until they meet the design requirements. The abductive methods emerge under the need for designing origami structures to satisfy geometric and functional requirements instead of accurately defined folded shapes. An example of such design problems is the design problem proposed in the first chapter. As shown in Figure 6(b), the typical abductive design methods have to go through iteration before the derivation of the final design. In each iteration, an abductive method adapts, simulates and rates each candidate solution. The advantages of the abductive design methods include:

1) They enable the search of a solution space, in which designers have no clear domain knowledge.
2) They broaden the types of design requirements to any one that can be computationally assessed and rated.
3) They are compatible for structure design considerations. So in the abductive methods, the material and 3D dimensions can also be designed if required.

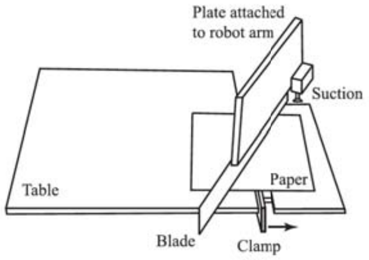
The limitations of the deductive design methods are from the difficulty of simulating a complex origami structure and the time cost of search solutions in a large and nonlinear design space, which is very common for origami design problem. In this dissertation, I will study applicable abductive origami design methods for a problem as stated in chapter I. My research will mainly focus on the geometric representation, the algorithm of emulating and evaluating an arbitrary design candidate, and the measures of accelerating the solution search.

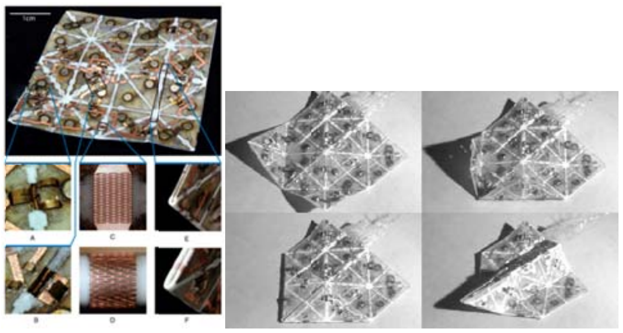
Nevertheless, for both type of origami design methods, if the crease pattern includes tessellations, the complexity of design problem will be mitigated. The design will re-target at each tessellated tile that can be defined by only a few design parameters. The study of tessellation pattern design is mainly on identifying the geometric influence on specific properties, such as mechanical performance, feasibility, efficiency, cost and a host of application dependent secondary functionality. Some of the studies on tessellation design are written in (Bateman 2001, Stroble, Nagel, Stone and McAdams 2010, Tachi 2010, Wang and Chen 2010). The lesson for this dissertation from the tessellation design research is its methodology of using geometric modularization to reduce the number of design parameters, thus simplifying the problem.

Intelligent Origami Structures

With the development of machines and robotics, automation has also been introduced into origami folding within the last few decades. With current technology, two realizable measures have emerged for automatically making physical folding of origami, which are robotic origami folding and self-folding origami. Robotic origami folding (Balkcom and Mason 2008, Van Den Berg, Miller, Goldberg and Abbeel 2011, Yukokoji, Tanaka and Kamotani 2006) is not the focus here. Robotic origami indicates the use of a robot or intelligent machine to fold origami sheets (Figure 7(a)). Although the origami engineers ultimately pursue a highly intelligent material that could achieve spontaneous folding motions by itself while being made into a sheet, with state-of-the-art self-folding origami systems still have to be integrations of several subsystems. These subsystems might include the folding actuators, a motion controller, the surface material, folding joints (creases), and a power supply.

The actual composition of any self-folding origami system differs according to its potential application. To realize repeatable folding and deployment motions on pre-determined creases as for deployable solar panels, antenna deflectors (Miura 2006, Miura 2009) and the programmable matter self-folding sheet (Hawkes, An, Benbernou, Tanaka et al. 2010), the surface is partitioned into polygons that are connected by crease joints. For solar panels or antenna deflectors, the actuators must have enough power, so they are usually motors or springs that are attached to the joints. For the smaller scaled programmable matter self-folding sheet shown in Figure 7(b), the SMA crease joints also serve as the folding actuators, and the folding motion of each crease joint is achieved by controlling its temperature change, which will cause the transition of the material's martensite phase and austenite phase. In both cases, an external motion controller is generally required to regulate the folding.





(a) (b)

Figure 7 Intelligent origami robots. (a) Origami folding robot by Devin et al. (b) Programmable matter robot by Hawkes et al.

Nevertheless, if the folding motion is designed to serve the manufacturing of a permanent curvature on the material, the external motion controller and power supply become redundant. In this situation, Skylar Tibbits (Tibbits 2012, Tibbits and Cheung 2012) proposed a solution called 4D printing. Based on 3D printing, 4D printing appends the fourth dimension of programmed folding motion onto the product. The folding motion is physically resulted to the expansion of some part of the smart material, while water is expected to act as an energy source.

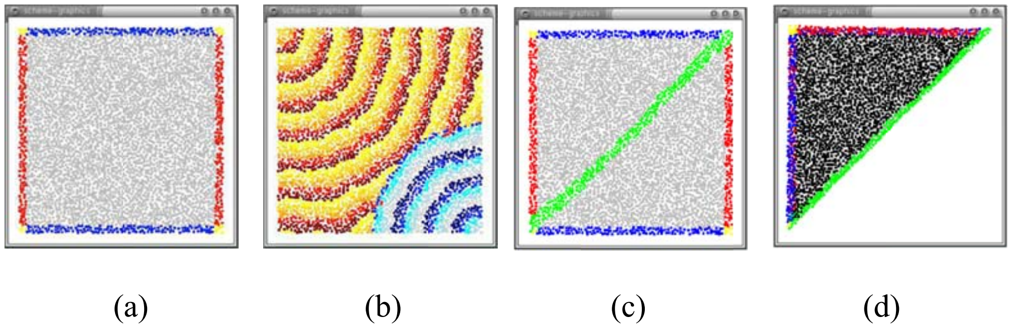


Figure 8 Self-regulated "epidermal" multicellular origami sheet and the stages for determining a fold by Nagpal et al. (a) Non-exited; (b) "Morphogen" propagation; (c) Identification of the cells with certain levels of morphogen density at the crease; (d) Folding the sheet along the crease.

If we say that both the programmable matter folding sheet and the 4D printed material are examples of self-folding on pre-determined creases, Nagpal et al., on the other hand, presented their theoretical solution for realizing arbitrary simple folds. They applied randomly distributed self-regulating "epidermal cells" on an origami sheet

(Figure 8) to determine creases (Nagpal 2002, Nagpal, Kondacs and Chang 2003, Stoy 2004). In the multicellular origami sheet, every cell dynamically emits and transfers types of chemicals. The cells that satisfy criteria on the concentrations of chemicals will be used to mark a fold (Huzita and Mitchell 1989). The cells that mark a desired crease will then taper themselves to form the fold. Although their research enables arbitrary creases to be realized in an artificial origami sheet, they didn't proceed to physically build their "epidermal" multicellular origami sheet to test their self-folding control.

The research on automated origami structure and folding has revealed new perspectives for origami design. The programmable matter origami innovates an origami design approach, which sets up pre-defined creases in an origami sheet and simplifies the problem to determining which pre-creases will be used in the final design. Moreover, The origami paper folding robot and the folding controlled by self-regulating cells are good instances for designing the designer. Instead of directly looking for creases, they design the systems that determine the folding.

Evolutionary Algorithm: Applications and Key Issues

Variations of evolutionary algorithms have been successfully implemented on system design problems. The application of EAs covers three stages of design procedure, including conceptual design, embodiment design and detailed design. Some recent applications that serve the conceptual design stage include automated design of room automation systems (Oezluek, Dibowski and Kabitzsch 2009), minimum weight compliant mechanism design (Sharma, Deb and Kishore 2013), MEMS bandpass filter design (Farnsworth, Benkhelifa, Tiwari and Zhu 2010), shelter design represented by the graphic grammar (O'Neill, McDermott, Mark Swafford, Byrne et al. 2010), multi-stage logistic network design (Lin, Gen and Wang 2009), building structure and floor blueprint design (Bollinger, Grohmann and Tessmann 2010), etc. The applications that serve the embodiment and detailed design stages include an iron-making system design for a blast furnace (Pettersson, Saxen and Deb 2009), a multiple-disk clutch design (Deb and Srinivasan 2006), the optimization of a leg mechanism (Deb and Tiwari 2005), sidelobe level reduction of a concentric circular antenna array (Mandal, Bhattacharjee and Ghoshal 2009), etc. The above structure design applications of EAs have already encompassed the industrial and research fields on manufacturing, robotics, architecture, as well as system management. Evolutionary algorithms are exceptional for their iterative solution search technic that is adapted for problems with non-linear and non-convex design search space.

To apply an evolutionary algorithm, there are three issues that must be solved with respect to the subject, objective and constraints of the specific design problem:

1) Choose a proper representation for the system;
2) Define an adequate objective function for estimating the performance of the candidate solutions;
3) Develop a set of efficient evolutionary operators (Kicinger, Arciszewski and De Jong 2005).

The representation for the system is mostly relevant to the computational issues including search efficiency and the mapping between a search space (genotypic space) and a space of actual designs (phenotypic space). A few criteria can be used to evaluate a representation. Primarily, the representation must ensure the mapping between the genotype and phenotype of a candidate solution is 1-to-1, which means that any genotypic encoding must correspond to a candidate solution, while any candidate solution must possess one genotypic encoding. Occasionally, an n-to-1 mapping is also acceptable, but the n-to-1 mapping may cause multiple global optima in the design search space. The second characteristic of a proper representation is continuity. Continuity requires that slight mutations on the encoding only cause small variations on the phenotypic features of the corresponding candidate solution. It is also recommended that the representation may not have a genetic code that represents an illegal candidate solution (illegal solution means that a genotype does not represent any phenotype for a given problem (Kicinger, Arciszewski and De Jong 2005)). The continuity does not need to be perfectly realized. Instead, reducing the number of bits in the genetic code, whose mutations will result in a significant change in the performance of the phenotype, is still a rule of thumb while we design the representation.

After forming a proper representation for the candidate solutions, it is necessary to formulate an objective function that adequately defines the mathematical model of the problem. The adequacy not only requires that the objective function must be able to comprehensively assess the candidate solutions according to the objective and constraints, but also implicates that the objective function must be an effective one resulting in apparent difference in fitness evaluations between good and bad candidate solutions.

The efficiency of an evolutionary method is directly controlled by the evolutionary operators. The evolutionary operators include selection, mutation, crossover (Schmitt 2001), and other complementary mechanisms. Usually mutation and crossover are also called genetic operators. The three basic operators - selection, mutation and crossover, which maintain the fundamental evolutionary procedure, emulate the survival, elimination, and reproduction of natural species. The design of basic evolutionary operators generally means choosing the proper variant for each of them and adapting the parameters. If the problem has a non-convex design space or multiple objectives, or if the evolution encounters slow finishing or pre-mature convergence, the evolutionary operators also need complementary mechanisms to guarantee the derivation of optimal solutions. Under the circumstance, researchers introduced co-evolution (Potter and De Jong 1994, Potter and De Jong 2000), external populations (Zitzler and Thiele 1999), Pareto ranking (Goldberg 1989), etc., to the evolutionary algorithms so as to preserve and balance the elitism and diversity within the evolving population.

Genetic Algorithm and Computational Evolutionary Embryogeny

Based on the desire of an abductive method to solve the design problem given in chapter I, this dissertation will implement genetic algorithm and another GA-base meta-heuristic, named computational evolutionary embryogeny, on the origami structure design.

GA, as a bio-inspired method, has been adapted and applied to many scientific and engineering areas. However, the studies have not come to a “panacea” version of GA. Instead, the users usually modify the GA parameters and add extra evolutionary operations in accordance with their own particular applications, so as to achieve both the elitism and diversity within the population of the candidate solutions.

Under the requirements of multiple properties of the design subject, the generic GA usually takes the weighted sum of the numerical estimations with respect to all the properties as the normalized fitness value for each individual (candidate solution in GA). The normalized fitness values are scalar, thus they can be easily used to compare and rank the individuals. The multi-objective GA, however, directly applies the vector of the evaluations on different properties without summing them up. The advantage of multi-objective GA is that it doesn’t require the users to clarify the priority ranking or weight factors on multiple properties. Lots of variations of multi-objective GAs have been proposed. Generally, these variations differ based on their fitness evaluation, elitism, and diversity preservation approaches (Konak, Coit and Smith 2006). Several other survey papers (Coello 1999, Fonseca and Fleming 1993, Fonseca and Fleming 1998, Lei and Shi 2004, Zitzler and Thiele 1999) have given summarizations of the multi-objective GAs on other perspectives. Multi-objective GAs usually apply the Pareto ranking (Goldberg 1989). To make sure that the multi-objective GA is able to perform more comprehensive searches through thorough exploration and exploitation within the solution space, several complementary measures are introduced to balance the elitism and diversity among candidate solutions. These measures include crowding distance

(Deb, Pratap, Agarwal and Meyarivan 2002), fitness sharing (Goldberg and Richardson 1987), pure elitist strategy (Konak and Smith 2002), and external population (Fieldsend, Everson and Singh 2003).

Computational evolutionary embryogeny (CEE) is a special case of genetic algorithm. The aspects that CEE is different from GA are the genetic representations for the candidate solutions, as well as how the genetic representation (a.k.a. genetic code, genotype) and physical representation (a.k.a. physical appearance, phenotype) of one solution candidate are related. In GA, an individual's physical representation is described explicitly by numerating and encoding its properties into a genetic code string (usually in binary), which is the individual's genetic representation. In CEE, an individual's genetic representation is also a string of genetic code, but the genetic code encrypts the set of genetic rules that instructs how its physical representation is constructed. The genetic rules play a similar instructive function as the DNA sequences in most living organisms. More clearly, in CEE, the genetic rules provide the guidelines for an individual to develop from an initial state (defined as the embryo) to its mature state, so as to disclose its physical representation. The way that the GA's genetic code describes an individual is defined as explicit coding, and the way that CEE's genetic code encrypting the instructive genetic rules as implicit coding. Nevertheless, for both CEE and GA, the design subject must be able to be fully parameterized, so that it can be described by a genetic code in either an explicit or an implicit way.

Several researchers have already attempted to introduce CEE into industrial design. Currently, most of them use this method in a classical manner to design simple and static structures, such as a planar pattern, or a support base. Peter Bentley, and Sanjeev Kumar were among the first ones providing a summarization of varieties of CEE (Bentley and Kumar 1999). They pointed out the features, advantages, and drawbacks of this method. They also attested that CEE provides benefits such as the reduction of search space, repetition, adaptation, and etc., while suffering higher difficulty to design and evolve. Chris Bowers used CEE method to generate 2D structures that duplicate or approximate colored patterns (Bowers 2005). He has also investigated the modularity of CEE (Bowers 2008), and further extended CEE to optimize artificial neural network structure . Or Yogev, Andrew A. Shapiro, and Erik K. Antonsson extended CEE for designing a 3D continuous inhomogeneous structure, which is essentially a simple support column (Yogev, Shapiro and Antonsson 2010). They formalized the artificial genetic codes that have the "if-conditional then-action" structure.

CHAPTER III

GENOTYPE-PHENOTYPE MAPPING*

In GA, any candidate solution is represented by the combination of its genotype and its phenotype. The genotype is also called the genetic representation in GA. The genotype of a candidate solution is usually defined by a binary string of 0s and 1s. For origami design, in order that a slight perturbation on the code won't result in significant change in the encoded features, Gray code replaces binary code. Gray code is an alternative binary numeral system where two successive values differ in only one bit (binary digit). The phenotype is the physical appearance of a candidate solution. It carries the geometric, topologic, and functional properties of an origami structure. For origami crease patterns, the phenotype is basically the geometric representation of all the vertices and creases. For 3D origami shapes, the phenotype needs to contain more information, such as the dihedral fold angles, the orientation, and the folding sequence, other than the crease pattern. Since the physical representation of a shape can be generalized and defined by a set of numerical parameters, the corresponding genotype –

*Part of this chapter is reprinted with permission from ASME, Journal of Journal of Computing and Information Science in Engineering, and IDETC/CIE 2013. Refer to: (1) McAdams, D. A. and Li, W. (2014). "A Novel Method to Design and Optimize Flat-Foldable Origami Structures Through a Genetic Algorithm." Journal of Computing and Information Science in Engineering **14**(3). To be printed in September 2014; (2) Li, W. and McAdams, D. A. (2013). "A Novel Pixelated Multicellular Representation for Origami Structures That Innovates Computational Design and Control." ASME 2013 International Design Engineering Technical Conferences (IDETC) and Computers and Information in Engineering Conference (CIE), Portland, OR.

the genetic code – can be easily determined by converting those parameters into Gray code by a desired sequence.

Therefore, the prerequisite condition of applying GA on origami design problems is to establish a suitable geometric representation, which includes the genotype, the phenotype, and a mapping between. A key requirement for a suitable geometric representation is that as the genotype mutates, the corresponding phenotype should not become invalid. In other words, any arbitrary sequence of 0s and 1s that consolidates into a genetic representation must be able to be mapped to a legal physical representation.

In established studies on origami engineering, the most applied geometric representation for computational methods is to use the two end points to denote the creases that define a crease pattern. And in most of the cases, the researchers focused more on the mathematics or the physical appearance of origami. Prior work has not emphasized on how a crease pattern or a folded shape is geometrically defined. However, for GA, the geometric representation and its syntax for genotypic encoding must be carefully designed.

In general, a suitable representation for GA method must satisfy the following minimal criteria:

1. Non-redundancy. Any representation points to only one crease pattern.
2. Completeness. Any arbitrary crease pattern possesses one representation.
3. Continuity & legality. A small perturbation in the representation results in a small change in the crease pattern (and won't cause the crease pattern to become illegal).

Established Geometric Representations of Origami Crease Patterns

The intuitive solution for forming a geometric representation of origami crease pattern is to list and define all the creases. Toward this direction, the mission of seeking for a geometric representation is simplified to deriving a method to define every crease.

The typical embedded representation for origami crease patterns defines all the creases by listing by their end points. For instance, a crease pattern shown in Figure 9(a) has four creases, thus can be represented by the set of 16 (4*4) arguments (A_1)

$$A_1 = \{0.5, 0.6, 0, 0.4, 0.5, 0.6, 0.2, 0, 0.5, 0.6, 1, 0.3, 0.5, 0.6, 0.4, 1\}$$

Among the set A_1, every four arguments define one crease, e.g. the first four arguments "*0.5, 0.6, 0, 0.4*" are the X and Y coordinates of the two end points of one crease. The corresponding genetic representation is derived by converting the argument set A_1 to Gray code. The genetic representation of the crease pattern based on the argument set A_1 is

C_1 = "*0111-0101-0000-0110-0111-0101-0011-0000-0111-0101-1111-0010-0111-0101-0110-1111*".

The embedded representation is complete for any arbitrary crease pattern, but it is extremely "fragile". Here, "fragile" means low non-redundancy, low continuity, and low legality, or say, any single bit of mutation in the genetic code will have a high possibility of causing the corresponding crease pattern to become invalid. When its genetic code for the crease pattern in Figure 9(a) mutates on the fourth bit, the first argument will change from 0.5 to 0.4, and the corresponding crease pattern will become invalid as shown in Figure 9(b). In Figure 9(b), the crease through the vertex (0, 0.4), which used to intersect with the other three creases at the vertex (0.5, 0.6), is now detached. Other than being practically fragile, the embedded representation has another problem, if it is applied for GA. In a crease pattern, an interior vertex is the common end point of several creases, or say, an interior vertex links several creases. But in the embedded representation, the linking relationship of the creases is not considered. This results in the definition of an interior vertices appearing in the representation several times. Therefore, if only one instance of an interior vertex changes, it will no longer match the other instances of the same vertex, and a crease will become detached to make the crease pattern illegal.

An alternative geometric representation, which is abstracted from Mitani's method of generating random crease patterns (Mitani 2011), eliminates the redundancy of interior vertex definition, so as to avoid the mutation on genetic coding resulting in the detachment among connected creases. In Mitani's research, he introduced an approach to generate random flat-foldable crease patterns. The approach creates a crease pattern in three steps – placing vertices, connecting pairs of vertices with creases, and then updating the vertices and creases to make the pattern flat-foldable. The alternative representation adopts the same procedure. But if the flat-foldability is not asked, the third step can be dropped. To define a same crease pattern as Figure 9(a), the alternative representation uses another set of arguments (A_2)

$$A_2 = \{0.5, 0.6, 0, 0.4, 0.2, 0, 1, 0.3, 0.4, 1, 1, 2, 1, 3, 1, 4, 1, 5\}$$

In A_2, the starting five pairs of arguments list the locations of the five vertices that are implicitly numbered by #1 to #5. Here, the #1 vertex is at (0.5, 0.6), #2 at (0, 0.4), #3 at

(0.2, 0), #4 at (1, 0.3), and #5 at (0.4, 1). The following eight arguments can be partitioned into four pairs (“1, 2”, “1, 3”, “1, 4” and “1, 5”), each of which indicates the two end points of a crease. With the four pairs of arguments, the topological formation of the crease pattern is uniquely determined. The equivalent genetic representation for the crease pattern that is defined by the argument set A_2 is

C_2 = “*0111-0111-0101-0000-0110-0011-0000-1111-0010-0110-1111-001-011-001-010-001-110-001-111*”.

In the alternative representation, mutations on the bits that define the coordinates of the vertices in the genetic code C_2 will only result in the dislocation of the vertices, thus won’t cause the invalidity of the crease pattern. However, the mutation of the bits that define the topological linking of the creases will cause the loss of creases or unwanted intersections. For example, mutation of the last bit of C_2 will alter the last argument in A_2 from 5 to 4. Therefore, the corresponding crease pattern Figure 9(c) of the mutated C_2 has two overlapping creases between the vertex #1 (0.5, 0.6) and vertex #4 (1, 0.3). It therefore becomes invalid, because a crease pattern may not have overlapping creases. Even if the two overlapping creases are treated as one, the crease pattern Figure 9(c) is not foldable, as it has a vertex (#1) that is connected with only three creases. If the mutation happens on the fifth bit from the last bit, the second argument to the last argument will change from 1 to 3. The resultant crease pattern Figure 9(d) is also invalid, because the original crease (“1, 5”) changes to (“3, 5”), and it intersects with the crease (“1, 2”).

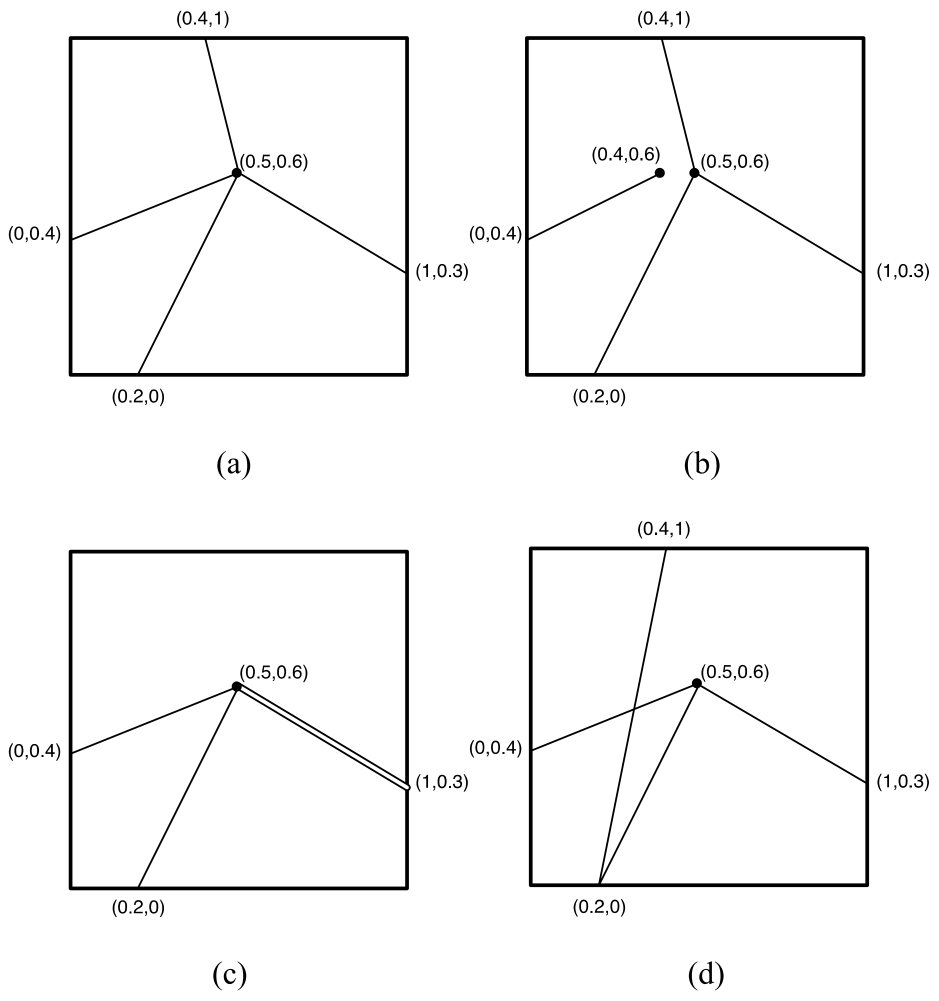


Figure 9 (a) A crease pattern represented by the locations of the vertices and the creases that link in between; (b) The mutation on one bit in the genetic code for the embedded representation of (a) results in disconnection of creases; (c) The mutation on the last bit in the genetic code for the alternative representation of (a) results in crease overlapping; (d) The mutation on the fifth bit from the last in the genetic code for the alternative representation of (a) results in crease intersection.

A common problem of the embedded and the alternative geometric representations is that arbitrary genetic codes do not always define a valid crease pattern. Compared to the embedded representation, the alternative representation is less “fragile”.

But the second group of arguments in the alternative representation is still very sensitive to genetic mutation. Thus, neither of the two representations is well suited for GA application. Therefore, it is necessary to develop new geometric representations, so that any random genetic representation can reflect a valid crease pattern, and any valid crease pattern doesn’t become invalid as its genetic representation mutates.

Ice-cracking – A Direct Representation

The embedded and alternative representations are probably the most intuitive ways for people with different levels of knowledge on origami to exchange or visualize well-designed crease patterns. According to the introduction so far, the alternative representation differs from the embedded representation by the removal of redundant definition of the shared vertices of multiple creases. However, it still failed to ensure the placement of creases being valid. The common shortcoming of the embedded representation and the alternative representation is their lack of concern on the sequence of vertices and creases being defined. What either of these two representations achieves is no more than generating a collection of points and links that ultimately represent vertices and creases. But they don’t include the mechanism to guarantee the legality of the placement of each origami topological component (vertex or crease).

If there is a GA-friendly geometric representation that defines a crease pattern by the vertices and creases, just as the embedded representation and the alternative representation do, it must ensure that there is no redundancy of origami element definition, and it must be able to preserve the legality of the crease pattern's topological structure under perturbations in the presentation.

At first glance, an origami crease pattern is topologically similar to a tree graph, where interior vertices are the nodes, border vertices are the leaves, and creases are the links. This analogous relation stands true for some of the crease patterns, such as the fish base variant in Figure 10(a). The representations of trees in GA has been studied in (Palmer and Kershenbaum 1994). Although the definition of a crease pattern is more than just about the topological structure, based on the results in (Palmer and Kershenbaum 1994), it is not very difficult to develop a way to also include the vertices' locations in the representation.

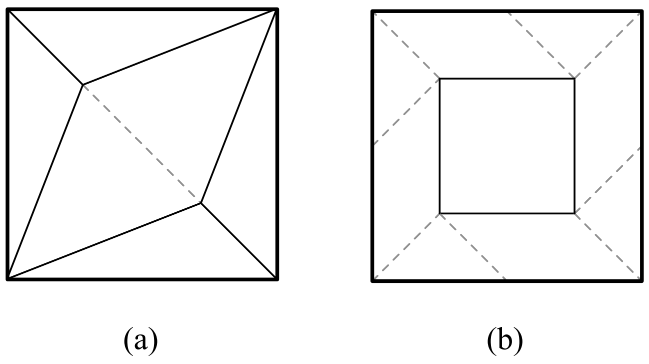


Figure 10 (a) The crease pattern of a fish base variation can be described as a tree graph; (b) The crease pattern of a pinwheel base is not a tree since there is a closed loop of creases.

But unfortunately, most of crease patterns are not trees. They will have closed loops of creases, such as the pinwheel base in Figure 10(b) that has four valleys forming a cycle. Another potential problem of using the tree graph representation for crease patterns is the same as the embedded representation and the alternative representation that the legality of the representation cannot be assured.

Based on the discussion of the three existing representations above, I develop a new geometric representation that can not only provide the completeness, but also satisfy non-redundancy and continuity. The basic idea to achieve it is to introduce the sequence of the vertices and creases, and to implicitly assign indices to each topological component defined in the representations.

Recall the traditional procedure of manually making an origami shape, we don't fold all the creases at once, even if we have the complete crease pattern. Instead, we partition the pattern into pieces and work on one of them at a time. For making each crease pattern piece, which is usually a single vertex fold or a simple fold, we still need to make some auxiliary folds to derive the real creases. Some of them are then completely or partially flattened in the completed crease pattern.

Another idea is to use a natural analogy of an origami crease pattern. If a blank and flat origami sheet can be taken as an untouched icy water surface, the creases and vertices can also be viewed as the cracks and crack forkings on it. Then with a heavy knock that will generate a break point on the surface, the initial cracks will propagate, and form new forkings and new cracks. In addition, combining the "divide & conquer" strategy used by traditional origami folding and the development of cracks in an icy water surface, I create a sequence that a new representation can apply to arrange and index the vertices and creases. Unlike the embedded representation placing creases without a specified sequence, or the alternative representation allocating all vertices prior to creases, the new representation defines one vertex in each step. And with the definition of each vertex, several dummy creases that intersect at the vertex will also be created in the same step. The dummy creases have a very similar function as the auxiliary folds in traditional origami folding procedure. They act as placeholders for real creases. The dummy creases must be placed at the right positions, so that every real crease must be the full length or a section of dummy crease, and each dummy crease is the placeholder for one and only one real crease. Based on this idea, the geometric representation is named “ice-cracking”.

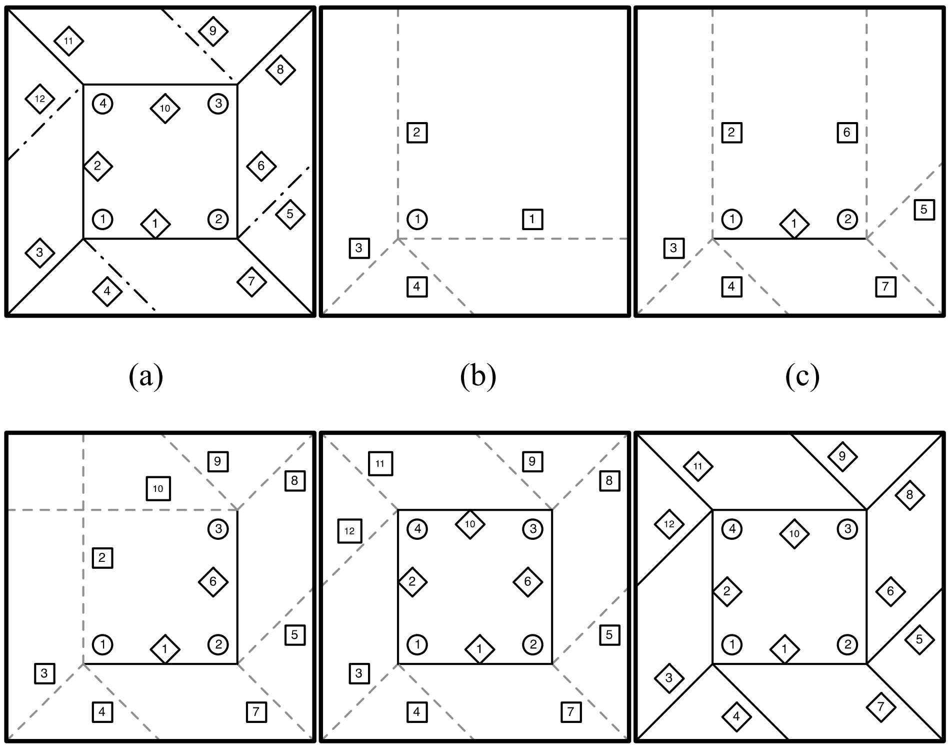
The “Ice-cracking” Representation of the Pinwheel Base

To form a pinwheel crease pattern as shown in Figure 11(a), where the 4 vertices are indexed by numbers in circles and the 12 creases by numbers in diamonds, without loss of generality, we (arbitrarily) choose the #1 vertex as the initial vertex to start the generation of the crease pattern by “ice-cracking”. For the #1 vertex, the number of creases (n_{NOC}) is four according to Figure 11(a). Four direction vectors then emanate from the #1 vertex. Here, the direction vectors are the dummy creases as mentioned earlier, and the #1 vertex is the origin vertex of the dummy creases. The number of direction vectors is equal to the number of creases that are supposed to intersect at the origin vertex #1. One crease will later coincide with each dummy crease, but the second end point (the first being the origin vertex #1) of each crease cannot be determined yet. In Figure 11(b), the existing dummy creases are displayed by dashed lines and labeled by numbers (1-4) in squares. The actions of specifying the starting vertex and dummy creases form the initialization step of the ice-cracking.

The next step is to establish a second vertex located on one of the existing dummy creases. Comparing Figure 11(a) and Figure 11(b), we find the #2 vertex on the #1 dummy crease, and the #4 vertex on the #2 dummy crease. The #2 and #4 vertices are both available to be the second vertex. In this case, I select the #2 vertex. As we locate the #2 vertex at the 2/3-distance point of the #1 dummy crease, the #1 crease that connects the #1 and #2 vertices is also determined. The n_{NOC} of #2 is also four, thus besides the known #1 crease, three other dummy creases are then created from the #2 vertex. These new dummy creases are numbered (5-7). The entire current pattern thus becomes Figure 11(c). The process of locating a new vertex on an existing dummy crease, together with the fixation of a new crease and the creation of new dummy creases, is called the forking step. If the dummy crease that the new vertex is supposed to be placed on during a forking step has intersections with other dummy creases, the new vertex cannot be placed beyond any of these intersections.

To form a pinwheel crease pattern as shown in Figure 11(a), where the 4 vertices are labeled by numbers in circles and the 12 creases by numbers in diamonds, without loss of generality, we (arbitrarily) choose the #1 vertex as the initial vertex to start the generation of the crease pattern by “ice-cracking”. For the #1 vertex, the number of creases (n_{NOC}) is four according to Figure 11(a). Four direction vectors then emanate from the #1 vertex. Here, the direction vectors are the dummy creases as mentioned earlier, and the #1 vertex the origin vertex of the dummy creases. The number of direction vectors is equal to the number of creases that are supposed to intersect at the origin vertex #1. One crease will later coincide with each dummy crease, but the second end point (the first being the origin vertex #1) of each crease cannot be determined yet. In Figure 11(b), the existing dummy creases are displayed by dashed lines and labeled by numbers (1-4) in squares. The actions of specifying the starting vertex and dummy creases form the ***initialization step*** of the ice-cracking sequence.

The next step is to establish a second vertex located on one of the existing dummy creases. Comparing Figure 11(a) and Figure 11(b), we find the #2 vertex on the #1 dummy crease, and the #4 vertex on the #2 dummy crease. The #2 and #4 vertices are both available to be the second vertex. In this case, I select the #2 vertex. As we locate the #2 vertex at the 2/3-distance point of the #1 dummy crease, the #1 crease that connects the #1 and #2 vertices is also determined. The n_{NOC} of #2 is also four, thus besides the known #1 crease, three other dummy creases are then created from #2 vertex. These new dummy creases are numbered (5-7). The entire current pattern thus becomes Figure 11(c). The process of locating a new vertex on an existing dummy crease, together with the fixation of a new crease and the creation of new dummy creases, is called the ***forking step***. If the dummy crease that the new vertex will be placed on during a forking step has intersections with other dummy creases, the new vertex cannot be placed beyond any of the intersections.



(a) (b) (c)

(d) (e) (f)

Figure 11 The steps taken by "ice-cracking" to derive the vertices and creases through a systematic sequence. (a) The full crease pattern with circled numbers labeling the vertices and diamonded numbers labeling the creases; (b) The initialization step that locates the first vertex; (c) The first forking step that gets the #2 vertex as well as the #1 crease; (d) The second forking step that gets the #3 vertex and the #6 crease; (e) The resolution step that gets the #4 vertex and two creases; (f) The resultant crease pattern without the MV-assignment after the finalization step.

At this stage, either the #3 or #4 vertex can be determined through a second forking step. We thus choose to create the #3 vertex, and convert the #6 dummy crease to the #6 crease. The n_{NOC} of #3 vertex is four, and its dummy creases #8 through #10 are created as well. Hence, we have derived the pattern as shown in Figure 11(d).

In Figure 11(d), #2 and #10 dummy creases have an intersection. In this step, we will locate the #4 vertex at this intersection. With the allocation of the new vertex #4, two creases are also determined simultaneously. The #2 crease is established as the section of the #2 dummy crease between the #1 and #4 vertices, and the #10 crease is established as the section of the #10 dummy crease between the #3 and #4 vertices. The $n_{\mathrm{NOC}}$of the #4 vertex is four. Since the #4 vertex has already had two definitive creases (#2 and #10), we only need to create two more dummy creases. The process of generating a new vertex at an intersection of dummy creases, together with the final fixation of two new creases and the creation of the new vertex's dummy creases, is named the ***resolution step***. The resultant pattern of this resolution step is shown in Figure 11(e).

All the vertices have been created through one initialization step, two forking steps, and one resolution step. We finalize the pattern by converting all of the remaining dummy creases to creases. This last step for obtaining a complete crease pattern is called the ***finalization step***. In the example case of the crease pattern in Figure 11, the finalization step will derive all the creases of Figure 11(f) without MV-assignment.

To determine the MV-assignment of the crease arrangement as shown in Figure 11(f), we need to first clarify the foldability requirements. As for the flat-foldability, we are able to use the method given in (Li and McAdams 2013). However, for other specific foldability requirements, such as orthogonal folding, we need to refine the mathematical and geometric rules and restrictions for arranging the vertices and creases, and the MV-assignment.

A Formal Instruction of the "Ice-cracking" Representation and the Syntax for Genotype-Phenotype Mapping

The example of the pinwheel base illustrates one possible sequence of developing all the vertices and creases of the pinwheel crease pattern through "ice-cracking". For any arbitrary crease pattern, the "ice-cracking" procedure can be plotted as shown in Figure 12. "Ice-cracking" starts with an initialization step that locates the first vertex. The initialization step is often followed by the first forking step to get the second vertex. After the first forking step, other forking steps and resolution steps can be arranged in arbitrary order, until the very last finalization step terminates the "ice-cracking" procedure. However, if the crease pattern has only one vertex, the initialization step will be immediately followed by the finalization step. A resolution step cannot be arranged right after the initialization step, since the dummy creases determined after the initialization step have no intersection other than the first vertex.

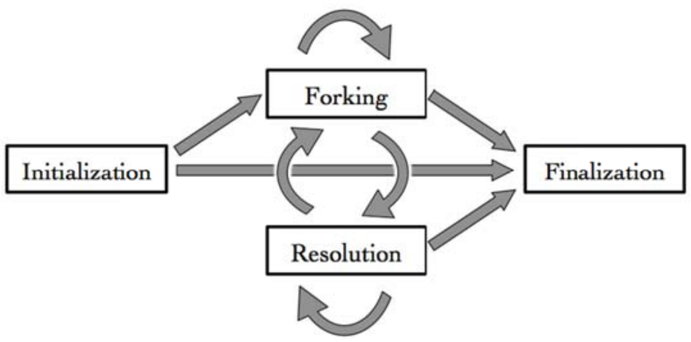


Figure 12 The four steps – initialization, forking, resolution and finalization – of "ice-cracking". The arrows show how the different steps can be arranged, with initialization being the first step and finalization the last step.

Table 1 shows the details of the four steps of "ice-cracking", together with the means that are used to encode the operations of each step into Gray binary code for the GA. In general, each step starts with a distinctive two-bit starting code as shown in Table 1. The following Gray code bits will encode the operations of the step in a fixed format.

Table 1 The operations in each step, and the arguments defining the operations

Step / starting code	Operations	Arguments for each operation
Initialization Step / 00	1. Locate the first vertex;	The x and y coordinate of the vertex - (x_1, y_1)
	2. Determine the vertex's n_{NOC};	n_{NOC}
	3. Provide the absolute angle of one dummy crease;	θ
	4. Draw the other dummy creases.	$\kappa_i (i = 1, \cdots, n_{\mathrm{NOC}})$
Forking Step / 01	1. Pick the dummy crease for the new vertex;	n_{DC}
	2. Locate the new vertex;	λ
	3. Fix a new crease;	
	4. Determine the vertex's n_{NOC};	n_{NOC}
	5. Draw the dummy creases.	$\kappa_i (i = 1, \cdots, n_{\mathrm{NOC}})$
Resolution Step / 10	1. Locate the new vertex at the location of one intersection of two dummy creases;	n_{VI}
	2. Fix two new creases;	
	3. Determine the vertex's n_{NOC};	n_{NOC}
	4. Draw the dummy creases.	$\kappa_i (i = 1, \cdots, n_{\mathrm{NOC}})$
Finalization Step / 11	1. Eliminate the intersections of dummy creases;	Fixed implicit argument set $\{n_{VI}=1, n_{\mathrm{NOC}}=4, \kappa_1=255, \kappa_2=255, \kappa_3=255\}$
	2. Convert dummy creases to creases;	
	3. Terminate the "ice-cracking".	

In the rest of this section, the four steps of "ice-cracking" are explained in detail along with the operations that are done in each step. The approach of encoding each step into genetic code is presented as well. In this research, the Gray code uses the M-QAM modulation, which has the same features with but is different from the natural Gray code translation (MathWorks 2014). The conversion between the Gray code, which is the genetic representation, and the parameters that defines a crease pattern is called the ***genotype-phenotype mapping***.

Initialization Step. Using "ice-cracking", any crease pattern must have one and only one initialization step. Two main operations are performed in the initialization step: locating the first vertex and determining the subsequent dummy creases. As shown in Table 1, the allocation of the first vertex requires specification of the x and y-coordinates. The arguments that define the $\mathrm{n}_{\mathrm{NOC}}$ and the angles of the dummy creases follow the allocation of the vertex. In general, if the first vertex has a $\mathrm{n}_{\mathrm{NOC}}$ of n, the initialization step will require n+3 arguments to define. The number of bits of Gray code for each argument is determined according to the required precision of the corresponding argument.

The initialization step for the pinwheel pattern Figure 11(b) serves to illustrate the encoding. Using a 1-by-1 square origami sheet, we set up a Cartesian coordinate system with the origin at the lower left corner of the sheet. Thus the #1 vertex is at (x_1 = 0.25, y_1= 0.25). If we define the resolution of the blank origami sheet to be 256×256 (28×28), both arguments representing the x-coordinate and the y-coordinate require 8 bits of Gray code. Therefore, x_1=0.25 is represented by ‘01110000’, while y_1=0.25 is also represented by ‘01110000’.

Then, several bits are used to represent the n_{NOC}. For instance, we could have a range of 4≤n_{NOC}≤11. In this case, 3 bits are needed to define the n_{NOC} among the 8(=2^3) possible values. However, if the crease pattern is constrained to be flat-foldable, the number of n_{NOC} for each vertex must be even according to Maekawa-Justin Theorem (Demaine and O'Rourke 2007). However, the n_{NOC} doesn’t have to be restricted to a range of consecutive integer values. A flat-foldable vertex cannot have odd n_{NOC}, thus we can use a pool of even numbers to give the available values of its n_{NOC}, such as {4,6,8 or 12} that requires only 2 bits in the genetic representation.

If the n_{NOC} is determined to be 4, 4 or 5(=4+1) additional arguments represent the direction vector angles of the dummy creases with respect to its foldability. In most situations, we set the x-positive direction of the Cartesian coordinate system to be the *0-rads line*. Then we pick one of the dummy creases to be the *baseline*, and an argument θ is used to define the absolute angle of the baseline w.r.t. the 0-rad line. Here, the #1 dummy crease becomes the baseline, and its absolute angle w.r.t. the 0-rad line is 0. The following 3 or 4 arguments will be used to define all 4 of the dummy creases. If the vertex has no foldability requirements, 3 arguments are enough to represent the absolute angles for the remaining 3 dummy creases in the same manner that the baseline dummy crease is defined. On the other hand, if the vertex needs to be flat-foldable, 4 more arguments are required. The one extra argument does not cause redundancy, but provides additional constraint for the three dummy creases due to the flat-foldability.

According to the Kawasaki-Justin theorem, the angles α_i between dummy creases should satisfy

$$\sum_{i \in odd} \alpha_i = \sum_{i \in even} \alpha_i = \pi \tag{4}$$

Hence for Figure 13(a), $\alpha_1 + \alpha_3 = \alpha_2 + \alpha_4 = \pi$. Then if the 4 arguments are $\kappa_i (i = 1,2,3,4)$, each of which is the decimal value of an 8-bit Gray code, the angles α_i are

$$\alpha_i = \begin{cases} \pi \times \frac{\kappa_i}{\sum_{j \in odd} \kappa_j} & (i\ is\ odd) \\ \pi \times \frac{\kappa_i}{\sum_{j \in even} \kappa_j} & (i\ is\ even) \end{cases} \tag{5}$$

In the end of the initialization step, all the newly created vertex and dummy creases are implicitly indexed for the following “ice-cracking” steps. In all, the entire initialization step for the flat-foldable #1 vertex of pinwheel pattern could use one possible argument set {x_1=0.25, y_1=0.25, n_{NOC}=4, θ=0, κ_1=255, κ_2=255, κ_3=255, κ_4=85}, which is then encoded by: ‘00 01110000 01110000 00 00000000 10101010 10101010 10101010 01100110’.

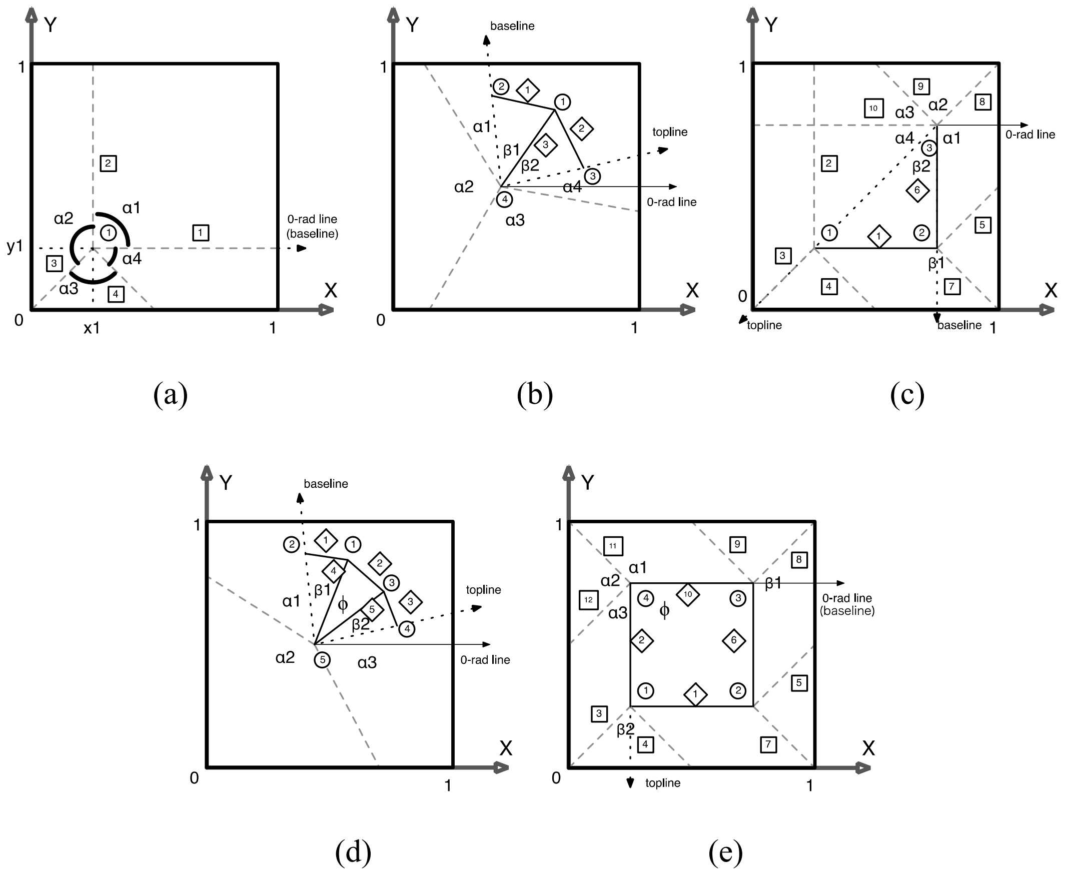


Figure 13 Illustrative crease patterns for "ice-cracking". (a) Initialization step; (b) Forking step; (c) The second forking step for getting a pinwheel pattern; (d) Resolution step; (e) The resolution step for getting the #4 vertex in a pinwheel pattern.

Forking Step. In each forking step, a new vertex is located on an existing dummy crease. Two arguments are used to give the location of this new vertex. The first argument $\mathrm{n_{DC}}$ indicates that the new vertex is on the $\mathrm{n_{DC}}$-th dummy crease within an ordered list of all existing ones. We use an 8-bit Gray code that is translated from the value of the argument $\mathrm{n_{DC}}$. If $\mathrm{n_{DC}}$ exceeds the total number $\mathrm{N_{DC}}$ of the existing dummy creases, we instead use the remainder of $\mathrm{n_{DC}}$ divided by $\mathrm{N_{DC}}$. The second argument λ, which also takes 8 bits in this study, defines the location of the new vertex on the dummy crease. If the distance from the origin vertex of the dummy crease and the dummy crease's nearest intersection with other existing dummy creases or with the origami sheet margin to the origin vertex is l, the distance d from the origin vertex to the new vertex can be derived as

$$d = l \times \frac{\lambda}{2^8} \tag{6}$$

After defining the new vertex, a new #3 crease linking the origin vertex and the new vertex is also fixed.

A third argument represents the n_{NOC} of the new vertex in the same way as in the initialization step. Suppose that in Figure 13(b) the #4 vertex is the new vertex determined in a prior forking step. If we already know that the n_{NOC} of the #4 vertex is 4, we will need another 3 or 4 arguments to define the 3 dummy creases. For any arbitrary origami crease pattern without foldability requirements (such as flat-foldability, rigid-foldability, orthogonal folding, etc.), 3 arguments are sufficient to define the angles of the 3 dummy creases; otherwise, extra arguments are needed to describe the constraints among the angle values of dummy creases that were introduced by the foldability requirement.

For a flat-foldable crease pattern, 4 arguments are required. As shown in Figure 13(b), a baseline vector and a topline vector that both start from the new vertex are needed. The baseline and the topline are determined, so that if a vector starts rotating from the position of the baseline around the new vertex counter-clockwise, it won't have any intersection with existing creases until it overlaps with the topline. If the vector rotates from the position of the topline counter-clockwise, however, it will have intersections with existing creases until it touches the baseline. We then define β1 as the angle between the #3 crease and the baseline, while β2 is the angle between the #3 crease and the topline. Next, we locate the dummy creases. Again, the newly created dummy may not have an intersection with any existing creases. Let α1 to α4 represent the angles among the baseline, topline and new dummy creases as shown in Figure 13(b). According to Kawasaki-Justin theorem, we have

$$\beta_1 + \sum_{i \in odd} \alpha_i = \beta_2 + \sum_{i \in even} \alpha_i = \pi \tag{7}$$

Thus if the 4 arguments are also $\kappa_i (i = 1,2,3,4)$, each of which is the decimal value of an 8-bit Gray code, the angles α_i are thus

$$\alpha_i = \begin{cases} (\pi - \beta_1) \times \frac{\kappa_i}{\sum_{j \in odd} \kappa_j} & (i\ is\ odd) \\ (\pi - \beta_2) \times \frac{\kappa_i}{\sum_{j \in even} \kappa_j} & (i\ is\ even) \end{cases} \tag{8}$$

In the end of the initialization step, all the newly created vertex, creases and dummy creases are implicitly indexed respectively. New indices follow the ones that indicate the existing topological components. In the same way, if we consider the derivation of the

#3 vertex in the pinwheel pattern (Figure 13(c)), the whole forking step could use one possible argument set of $\{n_{DC}$=6, λ =0.6667, n_{NOC}=4, κ_1=255, κ_2=254, κ_3=85, κ_4=127$\}$, and the Gray code thus is: ‘01 00000100 01100110 00 10101010 10101011 01100110 01011010’.

Resolution Step. The resolution step is similar to the forking step except for where the new vertex is located. In the resolution step, the new vertex must be created at an intersection between $\mathrm{n_I}(\geq 2)$ existing dummy creases. The first argument $\mathrm{n_{VI}}$ indicates which intersection is used. One implicit operation should be done initially to get a list of all the valid intersections between all dummy creases. A valid intersection is identified so that within any of the $\mathrm{n_I}$ intersecting dummy creases, this intersection is the nearest one from the corresponding origin vertex. To determine the argument $\mathrm{n_{DC}}$ in forking step, we pick the $\mathrm{n_{VI}}$-th valid intersection for the new vertex. Similarly, if $\mathrm{n_{VI}}$ is larger than the total number $\mathrm{N_{VI}}$ of valid intersections, we use the residue of $\mathrm{n_{VI}}$ divided by $\mathrm{N_{VI}}$. Right after the allocation of the new vertex, $\mathrm{n_I}$ new creases can also be fixed in this resolution step.

The second argument for resolution step is the n_{NOC}. In the example case of Figure 13(d), the #5 vertex is the new vertex based on creases #4 and #5. If we set the n_{NOC} to be 4, only two more dummy creases for this vertex need to be determined. Therefore, we will need 2 or 3(=2+1) more arguments. If the crease pattern has no explicit foldability requirements, 2 more arguments are used to define the dummy creases. However, if the crease pattern needs to be flat-foldable, we are going to need 3 more arguments. After defining the baseline and the topline in the same way as used in forking step, we define the angles β1 (the angle between #4 crease and the baseline), β2 (the angle between #5 crease and the topline), and ϕ (the angle between #4 crease and #5 crease). Then we let α1 to α3 represent the angles between the baseline, topline and new dummy creases as shown in Figure 13(d). According to the Kawasaki-Justin theorem, we have

$$\beta_1 + \beta_2 + \sum_{i \in odd} \alpha_i = \phi + \sum_{i \in even} \alpha_i = \pi \tag{9}$$

Thus the if the 4 arguments are also $\kappa_i (i = 1,2,3,4)$, each of which is the decimal value of an 8-bit Gray code, the angles α_i are

$$\alpha_i = \begin{cases} (\pi - \beta_1 - \beta_2) \times \frac{\kappa_i}{\sum_{j \in odd} \kappa_j} & (i\ is\ odd) \\ (\pi - \phi) \times \frac{\kappa_i}{\sum_{j \in even} \kappa_j} & (i\ is\ even) \end{cases} \tag{10}$$

At the end of the initialization step, all the newly created vertex, creases and dummy creases are implicitly indexed respectively. New indices follow the ones that indicate the existing topological components. In the same way, if we consider the derivation of the #4 vertex of the pinwheel pattern as in Figure 13(e), the whole resolution step could use one possible arguments set of {n_{VI}=1, n_{NOC}=4, κ_1=255, κ_2=255, κ_3=85}, and the code will be: ’10 00 10101010 10101010 01100110’.

In some situations, $\mathrm{n_I}$ is larger than 2, which means the new vertex is located at the intersection point of 3 or more dummy creases. Hence, the number of new dummy creases is $n_{\mathrm{NOC}} - \mathrm{n_I}$, but not $n_{\mathrm{NOC}} - 2$.

Finalization Step. The finalization step is the last step of “ice-cracking”. There are two different cases for the finalization step. One case is when there is no intersection left among the existing dummy creases. In this case the finalization converts the remaining dummy creases to final creases to terminate "ice-cracking". The second case occurs when the existing dummy creases still have intersection(s). Since the creases in an origami crease pattern cannot have intersections, before converting the dummy creases, we repeatedly run resolution steps using a constant argument set of {n_{VI}=1, n_{NOC}=4, κ_1=255, κ_2=255, κ_3=255}, until no more intersections exist. The argument set is a default setting. If there are some special design requirements, such as "the n_{NOC} at each vertex must be 8", other values for the argument set can be chosen accordingly, as long as they do not change throughout the entire finalization step.

Summary of "Ice-cracking" Representation and Its Encoding

The "ice-cracking" method provides a systematic sequence to define the vertices and creases in a crease pattern. In "ice-cracking", a crease pattern can be formed with an arrangement of the four different steps, which are all encoded naturally in the GA formalism. The genetic code of a crease pattern is thus the combination of the Gray codes of all the "ice-cracking" steps.

An important advantage of the "ice-cracking" crease pattern representation scheme and its encoding is the non-redundancy, so that any randomly generated code can be decoded into a valid crease pattern. This feature benefits the implementation of GA in such a way that even after unpredictable GA operations (i.e., crossover and mutation), the Gray code for one crease pattern will not become un-decodable to an origami crease pattern. Referring to Table 1, mutation on genetic code bits, which represents the arguments other than the starting codes of different "ice-cracking" steps and the n_{NOC}, will only result in small displacements of the locations of vertices or the orientations of creases. Even if the genetic code mutation causes the starting codes or n_{NOC}'s to change, the result will be a big change on the corresponding crease pattern, but not the nullification of the genetic code.

Also keep in mind that the mapping between Gray code strings and (flat-foldable) origami crease patterns is n-to-1. Rearranging the order of vertices and creases developed through "ice-cracking" could result in a different Gray code. For instance, if the crease pattern has two vertices, say #1 and #2, we can either generate the #1 vertex in the initialization step and create the #2 vertex by a following forking step, or alternatively generate the #2 vertex in the initialization step and create the #1 vertex by the following forking step. For GA, the n-to-1 genotype-phenotype mapping acceptable but not desired. One measure to improve it to a 1-to-1 mapping is by guaranteeing that in each step of "ice-cracking" steps, the new vertex can only be on the right of existing vertices.

In addition, appending more bits to the complete Gray code of an origami crease pattern will not cause any change to its corresponding crease pattern, since all the bits that follow the genetic code piece that defines the finalization step are not translated.

The Pixelated Multicellular Representation – An Indirect Representation

The existing geometric representations – the embedded representation and the alternative representation – and the "ice-cracking" representation have one feature in common that they all describe the vertices and creases in the crease pattern. Other than the vertices and creases, the third topological component of origami is the faces, which fill in the blank spaces that are partitioned by the creases and the borders of the origami sheet. In rigid and flat folding, the creases are straight, thus the faces are polygons. We can also prove that the face polygons are convex if no cuts or bends are allowed (Tachi 2009). For an origami sheet with a legal crease pattern, the faces can be treated as the inverse of the creases. Therefore, defining the creases is equivalent to defining the faces, and is sufficient for defining the entire crease pattern. However, since the easiest way to define a face polygon is by its edges, which are just the creases in origami crease pattern, most of the research on origami geometry and mathematics so far only implement the creases to define a crease pattern.

Is there a way to describe the faces without involving the definition of the creases? Nagpal's study on bio-inspired self-assembling systems has given an primitive idea (Nagpal 2002, Nagpal, Kondacs and Chang 2003). It is possible to describe a polygon through fine partitioning of the sheet into tiny pieces and marking the pieces that belong to the polygon. This approach is similar to the basic idea of finite element analysis (FEA). FEA has three major stages, which are discretizing the structure into finite elements, solving each element, and assembling the solutions (Reddy 2005). For describing a convex face polygon, we can also start with partitioning, then define each element, and finally put elements together to derive the polygon. If this idea leads to a geometric presentation of an origami crease pattern, which is no more than a union of origami faces, we have answered three questions:

1. How to discretize or partition an origami sheet?

2. How do we parameterize the elements?
3. How do we relate a swarm of elements to face polygons?

In the following sections, I will introduce my solutions for these three questions, as well as how I link them to derive a geometric representation that is compatible with GA.

Origami Sheet Pixelization

As mentioned above, Nagpal's research has given a possible direction for discretizing an origami sheet. But in this research, I will make further considerations. Here, let's first introduce two concepts – hard discretization and soft (fuzzy) discretization. The *hard discretization* is used to partition the structure into fixed elements that have clear boundaries with their adjacent peers. On the other hand, *soft (fuzzy) discretization* derives elements that can dynamically change their size. Just like soap bubbles, if one bubble expands, it will push its neighbors away or cause them to shrink in girth. FEA applies hard discretization, since the governing equation of each element requires unambiguous boundary conditions. For representing a polygon region, the soft (fuzzy) discretization is more applicable. The most important reason is that, with retractable boundaries, it is possible to minimize the resultant phenotypic change caused by mutation or perturbation on the representation.

Based on the soft (fuzzy) discretization, we don't simply cut the origami sheet into tiny elements. Instead, we scatter cellular elements over this entire region. We call the representation that will be developed based on such type of pixelization ***the pixelated multicellular representation (PMR)***.

In the PMR that will be studied and used for this research, we arrange cells in a variant square grid, which is obtained through rotating the square grid by 45°. An example variant square grid that has nine cells along the diagonal line of a square sheet is shown in Figure 14(a). The way in which the cells are arranged is, however, not essential for the pixelated representation. A random distribution of cells as used in Nagpal's representation (Nagpal 2002, Nagpal, Kondacs and Chang 2003) is also acceptable, but an even distribution is advantageous in keeping the whole pattern uniform and ready for arbitrary crease patterns.

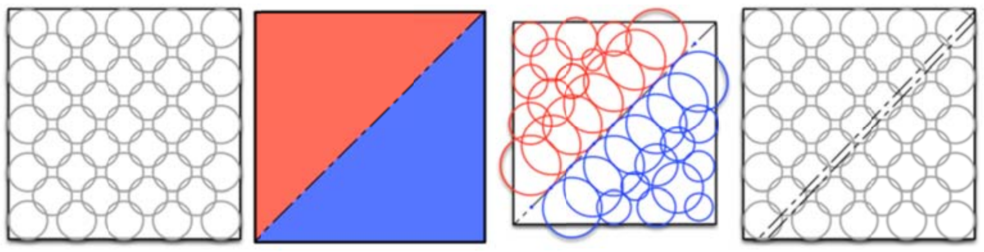

(a) (b) (c) (d)

Figure 14 Embryo in CEE. (a) An example pixelated multicellular pattern with 41 cells being arranged evenly; (b) A 2-color flat-foldable pattern with one diagonal crease; (c) A possible way to represent the crease pattern (b) with the pixelated pattern (a); (d) A crease pattern that cannot be represented by pixelated pattern in (a), because of too close creases.

The pixelated multicellular representation is structurally similar to putting an image on a LED matrix screen.

A cell can be thought of as one LED that sheds light on one soft (fuzzy) region that is located around the cell's center position. Each cell will possess three properties - cell type, cell size, and cell position. The cell position is determined by cell arrangement and resolution used for pixelization. The cell type indicates the cell's color property. The cell size is analogous to light intensity of an LED. The larger the cell size is, the higher light intensity the cell could "emit" onto its adjacent region. Meanwhile, the light intensity from a cell will gradually diminish along any direction that radiates from the cell's position.

Setup of the Cell Properties

For a pixelated multicellular representation, the three cell properties will determine the space of crease patterns that can be defined. We define the origami sheet to be 1 unit high and 1 unit wide.

Because of the 2-colorability of flat-foldable origami patterns, the number of cell types is two (additional cell types can be used to describe flat foldable structures, but Theorem 1 proves that two types are sufficient). The cell type of the i-th cell in the pattern is defined as $\mathrm{c_i}$.

$$c_i \in \{1, -1\} \tag{11}$$

where $c_i = 1$ means the *i*-th cell belongs to type 1 (blue), while $c_i = -1$ means the *i*-th cell belongs to type 2 (read).

Having specified the variant square grid used to arrange cells, the resolution of the grid determines the positions of the cells within the 1 by 1 origami sheet. The cell position of the i-th cell can always be expressed as

$$P_i = (x_i, y_i) \in \{\text{Variant square grid on a 1 by 1 sheet}\}, i = 0, 1, \cdots, p \qquad (12)$$

where p is the total number of cells.

In theory, the cell size can be any arbitrary value. Thus, to express the crease pattern in Figure 14(b) by the pixelated multicellular pattern Figure 14(a), the cell types and sizes are chosen as shown in Figure 14(c). In Figure 14(c), the two types of cells are visualized as blue and red circles. The blue cells suggest the region of the lower right face, and the red cells suggest the region of the upper left face. Each circle indicates the area of light illumination.

We define any cell that is positioned exactly on a crease to be an on-crease cell of the crease; and any cell whose distance to the crease is no more than the distance between two nearest cells to be a support cell of the crease. As a result, to specify the diagonal crease that separates the two faces in Figure 14(b), the on-crease cells will diminish to a size of 0, and the support cells will all be assigned to the size so that the corresponding circles could be tangent with the crease. Other cells in the sheet could have arbitrary size as long as their corresponding area of influence does not cross the crease. Figure 14(c) thus only provides one of the infinitely many ways to represent the crease pattern Figure 14(c) using the pixelated representation.

However the properties of the cells are assigned, the crease pattern shown in Figure 14(d) could not be represented based on the pixelated multicellular pattern Figure

14(a). The space between the two creases is too narrow, so no cell is located there. In the pixelated multicellular pattern, any face must contain at least one cell. Thus, we define the narrowest or smallest face the pixelated multicellular pattern can represent as the effective resolution of the pattern. Thus, if we wish the pixelated multicellular pattern to show crease pattern with tiny faces, we must increase the resolution of cells accordingly.

We discretize the cell sizes to facilitate computation. Valid cell sizes are evenly distributed discrete values between minimum and maximum admissible values. Thus the cell size s_i of i-th cell is

$$s_i = s_{min} + \frac{s_{max} - s_{min}}{n_p} \times n_i \ , \quad n_i \in \{0,1,2,\cdots,n_p\} \tag{13}$$

where s_{min} is the minimum size, s_{max} is the maximum size, and $n_p \in N$ is the number of possible values. Generally for variant square grid pixelated representation, s_{min} is usually chosen to be 0, and s_{max} is chosen to be equal to or a little bit larger than the distance between any cell and its second nearest neighbor.

Because of its discretization, the pixelated representation is not capable of defining the infinite space of arbitrary crease patterns. The theoretical number of patterns that can be defined is $(n_p \times 2)^p$.

A new problem arises when we want to define the two creases shown in Figure 14(d) separately. The creases are located so close to one another, the discrete pixelated representation may be the same for both creases. The effective precision is defined by the minimal change on determining the origami border, while tuning the cell sizes without changing their colors. The larger n_p is, the higher the effective precision will be.

The effective resolution and the effective precision determine the total representativeness of the pixelated multicellular pattern. Adding details in the pixelated pattern by increasing the resolution of cells and number of possible cell size values can result in the improvement on the effective resolution and the effective precision generally resulting in higher computational cost.

Understanding a PMR and Extracting the Crease Pattern Information

For the PMR as an equivalent representation for the crease pattern, we implement a so-called ***crease restoration algorithm (CRA)*** to extract the crease pattern. The outputs of the CRA should minimally include the crease arrangement, all possible MV-assignments, and all possible face overlapping orders.

The fundamental procedure of the CRA is to partition the whole sheet into different regions, and then to generate straight line-segments to divide the regions. We thus partition the sheet according to the distributions of the types of the cells. Ideally, we wish that each partitioned region on the sheet contains only cells of one type with as few outliers as possible. Because we also need the line segments to finally form an origami crease pattern, the lines ought to connect with each other end point by end point, and may not have any other intersections. Therefore, like the pixelated multicellular pattern intending to represent an origami crease pattern by coloring the faces, the essence of CRA is to identify the faces and their borders with regard to the coloration.

In mathematical expression, the CRA tries to solve a multi-objective optimization problem. We represent the 2D space that defines the flat sheet as S, and its border as Ω_S. The creases are C_i ($i = 1, 2, \cdots, m$), where m is the number of creases. The vertices are V_j ($j = 1, 2, \cdots, l$), where l is the number of vertices. The cells in the pixelated representation are $c_i = \{c_i, P_i, s_i\}(i = 1,2, \cdots, p)$, which are defined by their types, positions, and sizes. Thus the optimization problem is a search for a configuration of crease pattern that satisfies

Objective

$$\{\boldsymbol{C}_1, \boldsymbol{C}_2, \cdots, \boldsymbol{C}_m, m, \boldsymbol{V}_1, \boldsymbol{V}_2, \cdots, \boldsymbol{V}_l, l\}$$

$$= argmin \begin{cases} Err_{classification}(\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_p, \boldsymbol{C}_1, \boldsymbol{C}_2, \cdots, \boldsymbol{C}_m, m) \\ Err_{flat-foldability}(\boldsymbol{C}_1, \boldsymbol{C}_2, \cdots, \boldsymbol{C}_m, m, \boldsymbol{V}_1, \boldsymbol{V}_2, \cdots, \boldsymbol{V}_l, l) \end{cases}$$

Constraints

$$\begin{cases} \boldsymbol{C}_1, \boldsymbol{C}_2, \cdots, \boldsymbol{C}_m \subset \boldsymbol{S} \\ \boldsymbol{V}_1, \boldsymbol{V}_2, \cdots, \boldsymbol{V}_l \in \boldsymbol{S} \\ \boldsymbol{C}_i \cap \boldsymbol{C}_j = \emptyset \ or \ \boldsymbol{V}_k \ (i, j = 1, 2, \cdots, m \, ; k = 1, 2, \cdots, l) \\ N_k \geq 1, if \ \boldsymbol{V}_k \in \boldsymbol{\Omega_S} \\ N_k = 2t + 2, if \ \boldsymbol{V}_k \notin \boldsymbol{\Omega_S}, t = 1, 2, \cdots \end{cases} \tag{14}$$

where N_k is the number of creases that intersect at a vertex $\boldsymbol{V}_k$ (called interior vertex $if\ N_k \notin \boldsymbol{\Omega_S}$, or exterior vertex $if\ N_k \in \boldsymbol{\Omega_S}$); $Err_{classification}$ is the classification error according to the multicellular pattern and creases; $Err_{flat-foldability}$ is a quantified evaluation of the flat-foldability error, and its definition will be explained later in this section.

CRA has four stages - resampling, clustering, creasing, and finalization.

(1) Resampling. In some cases, cell resolution remains low based on computational restrictions. Therefore, we re-pick some data points as the proxies for the cells to facilitate the application of the algorithm used in the following clustering stage

of the CRA. The resampled data points need to be more densely distributed than the cells over the entire region. The reason why we require resampled data points will be explained as we introduce the clustering algorithm in Section 4.2.

Each resampled data point has three major properties, which are position, color, and color reliability. The positions of data points are already known as we resampled them. The color $\mathrm{O_i}$ and color reliability $\mathrm{R_i}$ of the i-th data point $\mathrm{o_i}$ are determined by the accumulated light intensity from its neighboring cells. The light intensity on data point $\mathrm{o_i}$ from its j-th neighboring cell is expressed as $\mathrm{I_{i,j}}$. In practice, we assume that the distribution of light intensity from any cell obeys either a triangular distribution or a normal distribution, which is parameterized by the cell size $\mathrm{s_j}$ and constants that describe the distribution.

For triangular distribution of cell's influence strength, we have

$$R_i = \sum_{j=1,\cdots,p}(-1)^t I_{i,j}$$

$$= \sum_{j=1,\cdots,p}(-1)^t \max\left(0,\frac{H}{G}\left(Gs_j - d_{i,j}\right)\right), t = \begin{cases} 0, if\ c_j = 1 \\ 1, if\ c_j = -1 \end{cases} \tag{15}$$

For normal distribution of cell's influence strength, we have

$$R_i = \sum_{j=1,\cdots,p}(-1)^t I_{i,j}$$

$$= \sum_{j=1,\cdots,p}(-1)^t H s_j e^{-Kd_{i,j}}, t = \begin{cases} 0, if\ c_j = 1 \\ 1, if\ c_j = -1 \end{cases} \tag{16}$$

where G, H and K are constant coefficients that describe the distributions, s_j is the size of j-th cell, and $d_{i,j}$ is the distance from the center of j-th cell to the data point o_i. We can determine that the color property of each data point to be

$$O_i = sign(R_i) = \begin{cases} 1\ (blue) \\ -1\ (red) \end{cases} \tag{17}$$

(2) Clustering. After resampling data points, we coarsely partition the regions for origami faces by clustering the data points. A region with a convex border that covers data points of a same color will come to a cluster, which will later represent a face in the crease pattern. Prior to clustering, we don't have the number of clusters. In this situation, density-based clustering could effectively detect the clusters, as well as the natural number of clusters (El-Sonbaty, Ismail and Farouk 2004). Ester et al. proposed an algorithm called DBSCAN (Density-Based Spatial Clustering of Applications with Noise) that can discover arbitrarily shaped clusters and handle noise (Ester, Kriegel, Sander and Xu 1996).

DBSCAN is a density-based clustering approach. Thus, for each point of a cluster within its "ϵ-neighborhood" for a given $\epsilon > 0$, the "density" has to exceed a certain threshold. DBSCAN can apply any different definitions of "neighborhood" and "density", as long that the "neighborhood" is a symmetric and reflexive binary predict, and the "density" calculation method can represent the "cardinality" of the defined neighborhood (Ester, Kriegel, Sander and Xu 1997). CRA applies DBSCAN, whose "ϵ-neighborhood" is simply distance based, that

$$N_\epsilon(o) = \{o' \in D | distance(o, o') \leq \epsilon, \epsilon > 0\} \tag{18}$$

where o and o' are two data points, D is the union of all data points, and "density" of a target data point is the summed count of its ϵ-neighbor data points (including the target data point) that also satisfy the cluster's criteria.

We define the cluster's criteria by a function

$$CR(o) = true, with\ o \in D$$

Several basic definitions are given below.

Definition 1. (*Core Data Point*) A data point o_i is defined as a core data point, if the union of its ϵ-neighbor data points, $N_\epsilon(o_i)$ (including o_i), satisfies a certain set of cluster's criteria CR. Therefore,

$$o_i\ is\ a\ core\ data\ point\ , iff. CR(o_j) = true, \forall o_j \in N_\epsilon(o_i)$$

Definition 2. (Directly Density-Reachability Matrix)

$$DDR(i,j) = \begin{cases} 1\ , if\ o_i \in N_\epsilon(o_j), CR(o_i) = true \\ \ , and\ o_j\ is\ a\ core\ data\ point \\ \quad 0, otherwise. \end{cases} \tag{19}$$

$DDR(i,j) = 1$ means that data point o_j is directly density-reachable from data point o_i.

Definition 3. (Reversible Directly Density-Reachability Matrix)

$$RDDR(i,j) = \begin{cases} 1\ , if\ DDR(i,j) = DDR(j,i) = 1 \\ \quad 0, otherwise. \end{cases} \tag{20}$$

$RDDR(i,j) = 1$ means that data point o_i and data point o_j are directly density-reachable with each other.

Definition 4. (Density-Reachability Matrix)

$$DR(i,j) = \begin{cases} 1\,, if\ there\ is\ a\ sequence\ o_i, o'_{k_1}, o'_{k_2}, \cdots, o'_{k_p}, o_j, p = 0,1,\cdots \\ where\ DDR(i,k_1) = DDR(k_1,k_2) = \\ \cdots = DDR(k_{p-1},k_p) = DDR(k_p,j) = 1 \\ 0, otherwise. \end{cases} \tag{21}$$

$DR(i,j) = 1$ means that data point o_j is *density-reachable* from data point o_i.

Definition 5. (Reversible Density-Reachability Matrix)

$$RDR(i,j) = \begin{cases} 1\,, if\ DR(i,j) = DR(j,i) = 1 \\ 0, otherwise. \end{cases} \tag{22}$$

$RDR(i,j) = 1$ means that data point o_i and data point o_j are density-reachable with each other.

Definition 6. (Density-Connectivity Matrix)

$$DC(i,j) = \begin{cases} 1\,, if\ there\ exists\ a\ data\ point\ o_k \\ that\ DR(k,i) = DR(k,j) = 1 \\ 0, otherwise. \end{cases} \tag{23}$$

$DC(i,j) = 1$ means that data point o_i and data point o_j are *density-connected*.

Definition 7. (Clustering and noise) A clustering $CLUSTERING_D$ of D w.r.t. N_ϵ and CR_{CDP} is the union of all density-connected sets w.r.t. N_ϵ and CR_{CDP} that can be found in D, $CLUSTERING_D = \{CL_1, \cdots, CL_n\}$. The noise is defined as $NOISE_D = D \backslash (CL_1 \cup \cdots \cup CL_n)$.

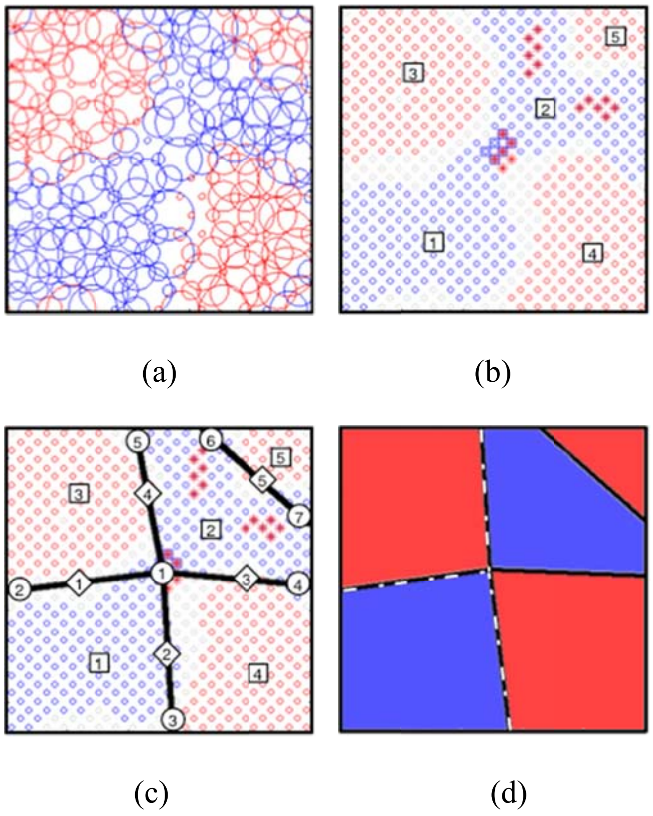


Figure 15 (a) A pixelated multicellular pattern; (b) Resampled data points and their clustering in crease pattern restoration algorithm; (c) The corresponding starting crease pattern with its vertices, creases and faces being numbered respectively by circular, diamond and square marks; (d) The final updated crease pattern with one flat-foldable MV-assignment.

Since the CRA clusters the resampled data points by their color reliabilities, it sets up a small threshold value of $\xi > 0$, so that a data point o_i will either fall to some blue cluster CL_j^b if $R_i \geq \xi$, or will fall to some red cluster CL_k^r if $R_i \leq -\xi$. Any other data point, whose color reliability has an absolute value of smaller than ξ, won't be used during clustering. With proper setting of N_ϵ and CR_{CDP}, the clusters $CLUSTERING_D$ will be obtained according to Definition 7. For the case of resampled data points in Figure

15(b) according to the pixelated multicellular pattern in Figure 15(a), blue and red circles are data points that are clustered into red and blue clusters, while grey ones are data points that are not clustered. In the entire sheet, there are totally two blue clusters (square marks 1 and 2), and three red clusters (square marks 3, 4 and 5).

Here, we need to explain the reason why we use the resampled data instead of the cells for clustering. Clustering is required before we proceed to the next stage of creasing, since the core problem that the DBSCAN clustering stage resolves is to dig out the natural number of clusters that is implicated in the pixelated multicellular pattern. But since we use LED matrix screen as the analogy of the pixelated multicellular pattern, the data points that represent what the whole colored pattern really look like are more proper for clustering than the cells that represent the light sources (LEDs). If the resolution of the cells is equivalent to the resolution of the LEDs, we can understand the resolution of the resampled data points as the resolution of eyes for perceiving the picture shown on the LED matrix screen. Moreover, data points of a higher resolution can more sensitively and accurately deal with the suspicious noises. In Figure 16, (a) is a pixelated multicellular pattern with two red cells that are surrounded by blue cells. According to most density-based clustering algorithms including DBSCAN, the two red cells will inevitably be identified as noise. But due to their relatively larger sizes, the two red cells should have the probability to stand out as a sole cluster.

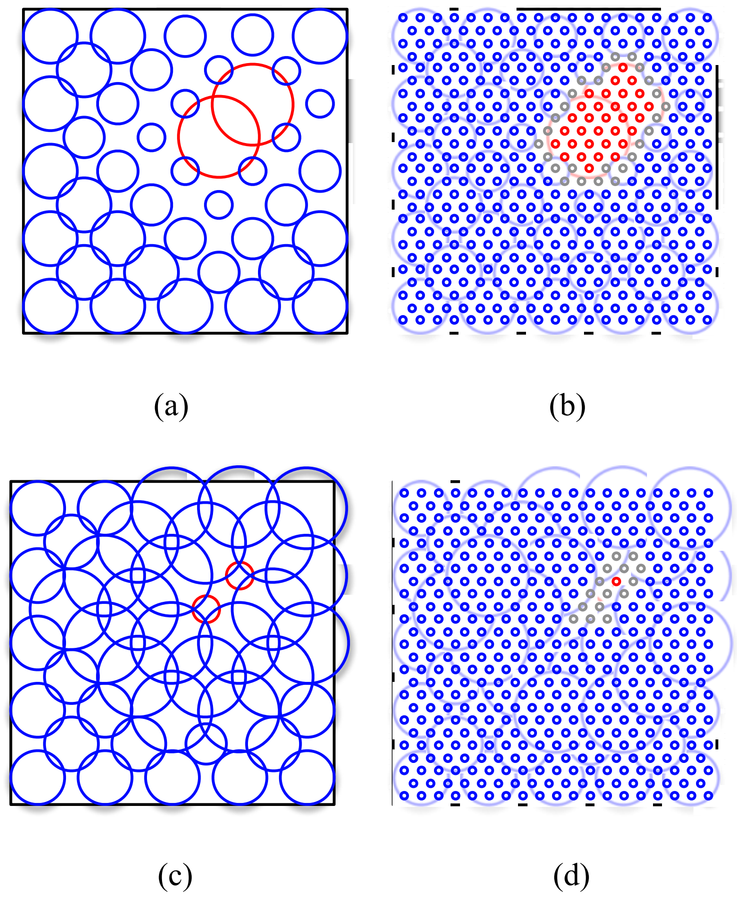


Figure 16 (a) A pixelated multicellular pattern, which has only two relatively larger red cells that are surrounded by blue cells; (b) Resampled data points according to (a); (c) Another pixelated multicellular pattern, which has only two relatively smaller red cells that are surrounded by larger blue cells; (d) Resampled data points according to (c).

Therefore, we get the resampled data points as shown Figure 16(b). The larger resolution of the resampling result in an enough number of red data points that approve the local region of the red cells being a cluster. Similarly, Figure 16(c) also has two red cells surrounded by blue cells, but sizes of the red cells are significantly smaller. The resampling of pattern Figure 16(c) results in Figure 16(d), that has only one red data point. Then the two red cells in Figure 16(c) cannot be a cluster, and must be identified as noises. It is also true that by further improving the resolution of the resampled data, we can anticipate the two red cells in Figure 16(c) to result in enough number of red data points to be a cluster. However, as our eyes having limited resolution, we must set up a resolution limit for resampling as well.

(3) Creasing. The next step is to place linear classifiers among clusters to separate them into the faces of the crease pattern. The region of a cluster and its surrounding linear classifiers become a face and its creases. The color of the face is the same as the color property of its corresponding data cluster.

There are several rigid demands on the classification. Firstly, every small sub-region in the origami sheet ought to be assigned to one of the faces. Secondly, each interior vertex must be placed within a constrained distance from the intersection of 4 or more clusters.

During classification, we need to minimize both the classification error and the flat-foldability error within the entire pattern. The classification error function $\mathrm{Err}_{\mathrm{classification}}$ applies the same definition used by most classification problems. With the color reliability acting as a weight for each data point, a more reliably blue or red cell that is misclassified will contribute more to the classification error.

$$Err_{classfication} = \{\sum R_i\,, if_{o_i}\ is\ miscallsified\} \tag{24}$$

The flat-foldability error function $\mathrm{Err}_{\mathrm{flat-foldability}}$ measures the least flat-foldable interior vertex in the pattern.

According to Kawasaki-Justin Theorem, we define and normalize the local flat-foldability error of each vertex as

$$Err_{flat-foldability,i} = \frac{\left|\alpha_{i,1}-\alpha_{i,2}+\cdots+\alpha_{i,k_i-1}-\alpha_{i,k_i}\right|}{2\pi} \tag{25}$$

where $\alpha_{i,1}, \alpha_{i,2}, \cdots, \alpha_{i,k_i-1}, \alpha_{i,k_i}$ are the consecutive angles between the creases around interior vertex $\boldsymbol{V}_i$, and k_i is the number of those angles. $Err_{flat-foldability,i}$ is 0 for perfectly flat-foldable vertex, and it can get a maximum value of 1. The flat-foldability error function will be

$$Err_{flat-foldability} = \max(Err_{flat-foldability,i} \mid \mathrm{V_i}\ is\ interior\ vertex) \tag{26}$$

The multi-objective minimization problem is non-linear. Therefore, an iterative search method could either periodically update the crease pattern as a whole, or only update one of the components in each cycle. Because both the error functions are formulated basing the local error function, it would make the problem easier if we optimize the local error function of one vertex at each time. We thus apply a *cascade direct search method*. The method is defined as a cascade because we apply a high level random search method that periodically updates the crease pattern, while in each period, we apply a *low level random jump search* to update only one vertex.

We must first give a definition of *saddle region* before we define the initial guess of the cascade search. A saddle region is the region where multiple blue clusters and multiple red clusters intersect. When getting the crease pattern, one interior vertex will take place in each saddle region. In Figure 15(b), the blue squares and red stars are data points that indicate suspect saddle regions. The actual saddle regions are those where blue squares and red stars aggregate together. Thus in this case, there is only one saddle region, which is located near the center of the sheet, and the only interior vertex will be placed in this saddle region.

To define a starting crease pattern of cascade direct search, we set one interior vertex at the center of every saddle region, and one crease to be the optimal classifier for each pair of neighboring data clusters. The exterior vertices will be sequentially decided as the intersection points of the creases with sheet margins. In Figure 15(c), which shows the corresponding starting crease pattern for Figure 15(b), vertices, creases and faces are numbered by circular, diamond and square marks respectively. As previously stated, the only interior vertex 1 is initially placed at the center of the saddle region. Creases 1-4 are the linear classifiers, which go through a fixed point at vertex 1, for cluster pairs among clusters 1-4. Crease 5 doesn't go through any interior vertices, so it only needs to optimally separate the two clusters 2 and 5. Exterior vertices 2-7 could then be attained as the intersection of creases and sheet margins. Be notified that if the origami structure is not confined by flat-foldability, the crease restoration algorithm shall terminate at this point.

Under the consideration of flat-foldability, the cascade direct search will tune the creases stepwise, until the minimization on both global classification error and global flat-foldability error is achieved. In the example, Figure 15(c) is finally updated to the one in Figure 15(d). This resulting crease pattern balances the two error functions globally for the whole pattern.

(4) Finalization. The finalization step mainly deals with optimizing values of the two error evaluations in equation (3.13) and (3.15). CRA also derives all the MV-assignments and face overlapping orders that pass the flat-foldability check through the procedure given in in Section 2.2.

The outcome of finalization is thus the crease pattern with its flat-foldable combinations of MV-assignment and face overlapping order. To favor broader applications, we could also calculate the flat-folded state profile, and number of overlapping face layers according to the crease pattern on demand as well. For the crease pattern in Figure 15(d), the flat-folded profile has an area of 0.3006 compared to the total area of 1 of the 1 by 1 origami sheet and a maximum of 3 layers of face overlaps.

Summary of PMR Representation and Its Encoding

In this section, I proposed a novel pixelated multicellular representation for origami structures. The PMR defines an origami structure by distributed cells. The cells fundamentally work as conceptual indicators, which directly define the origami by coloring the faces in the same way that LEDs in a LED matrix screen display a picture. Equivalent to an LED, each cell will have three key properties – cell type that shows the color, cell size that indicates the light intensity, and cell position that implies the resolution of cells. Every collection of cells with the same type (color) will define an origami face. The creases are linear classifiers generated to separate faces. I proposed a crease restoration algorithm to extract the equivalent crease pattern from a pixelated multicellular representation. I applied a special type of origamis that are flat-foldable to demonstrate CRA comprehensively.

The PMR is advantageous for its availability of expressing any arbitrary simple or non-simple crease, without losing the non-redundancy in representation. But after discretizing the cell properties for the computational applications, the representativeness of a pixelated multicellular pattern depend on the effective resolution and effective precision, which are the metrics relative to the setting of two of the cell properties – cell position and cell size. A third cell property is the cell color. In this research, the 2-colorability feature of flat-foldable origami has enabled the cell color to be chosen between minimally two colors.

The application of the PMR is more theoretically complicated and more computationally complex than ice-cracking, but it opens an interface for structural analysis techniques, such as FEA. To support such technique, the PMR can append the material properties or thickness value for each cell.

Summary on Two Designs of Geometric Representations

The two geometric applications (genotype–phenotype mappings) apply similar evolutionary algorithms, but they differ on the representation of candidate solutions. The quality of the representations can be assessed by how well they satisfy the four major requirements for designing good representations (Gen and Cheng 1999). The four major requirements include non-redundancy, completeness, legality, and continuity. In general, non-redundancy, completeness and legality demand the genotype-phenotype mapping

(G-to-P mapping) not to be 1-to-n, 0-to-1, or 1-to-0. Accordingly, the theoretically favorable G-to-P mapping is 1-to-1. However, an n-to-1 mapping is also acceptable, though it may cause one optimum in the phenotype space being mapped to multiple optima in the genotype space. Continuity requires that small variations in the genotype space due to mutation only cause small variations in the phenotype space.

In this research, the three applications invoke two types of genotype representations. The ice-cracking method applies the generative descriptions to define a crease pattern directly, while the PMR implements an indirect representation that equalizes a crease pattern and the equivalent artificial embryo form.

According to the comparison, the ice-cracking representation and the computational embryogeny both have good non-redundancy and completeness. However, computational embryogeny could not guarantee perfect legality, and the ice-cracking include a few bits (no more than 3%), whose mutation will cause substantial permutations on the crease pattern.

CHAPTER IV
GENETIC ALGORITHM AND COMPUTATIONAL EVOLUTIONARY EMBRYOGENY

The goal of this research is to solve the origami structure design problem that was described in Chapter I. The type of problem doesn't ask for achieving specific folded shape, but it will indirectly provide descriptions on geometric and functional features to help designers frame out their designs. In this type of design problem, without a clear target shape, it is impossible to make an assertion on the topological structure of the desirable design. Thus it is impossible to tell which deductive design method should be selected, since deductive methods mostly only serve one type of origami topology. As a result, it is inevitable that we have to expand the search space to include any arbitrary crease pattern, regardless its topological structure. And because of the fact that this type of design problem is NP-complete, an abductive design method, which features the procedure of first proposing and then proving, is necessary. This research selects the genetic algorithm and one of its variations, both of which implement the abductive problem solving logic.

Genetic algorithm (GA) is one search heuristic that is inspired by the process of natural evolution. It is generally used to derive an optimized solution through iteratively proposing, adapting, and selecting candidate solutions. The most noteworthy advantage of the genetic algorithm is that it avoids formulating an explicit model for the embedded design search space by constructing a substitutive search space defined by the genetic code. The fundamental framework of the genetic algorithm is shown in Figure 17. And the workflow chart based on the stages in each GA epoch provides an alternative description of the same procedure in Figure 18.

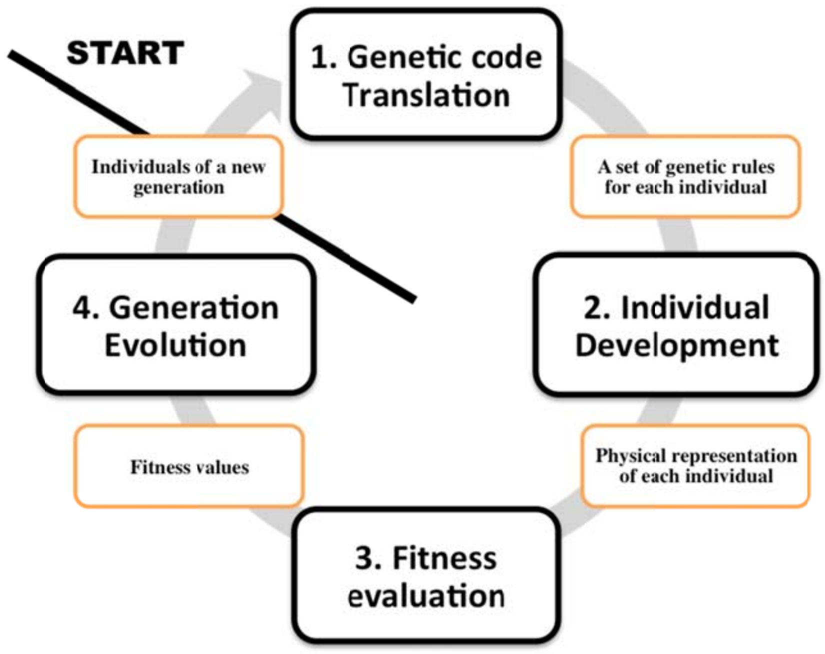

START
1. Genetic code Translation
A set of genetic rules for each individual
2. Individual Development
Physical representation of each individual
3. Fitness evaluation
Fitness values
4. Generation Evolution
Individuals of a new generation

Figure 17 An operational period cycle of GA

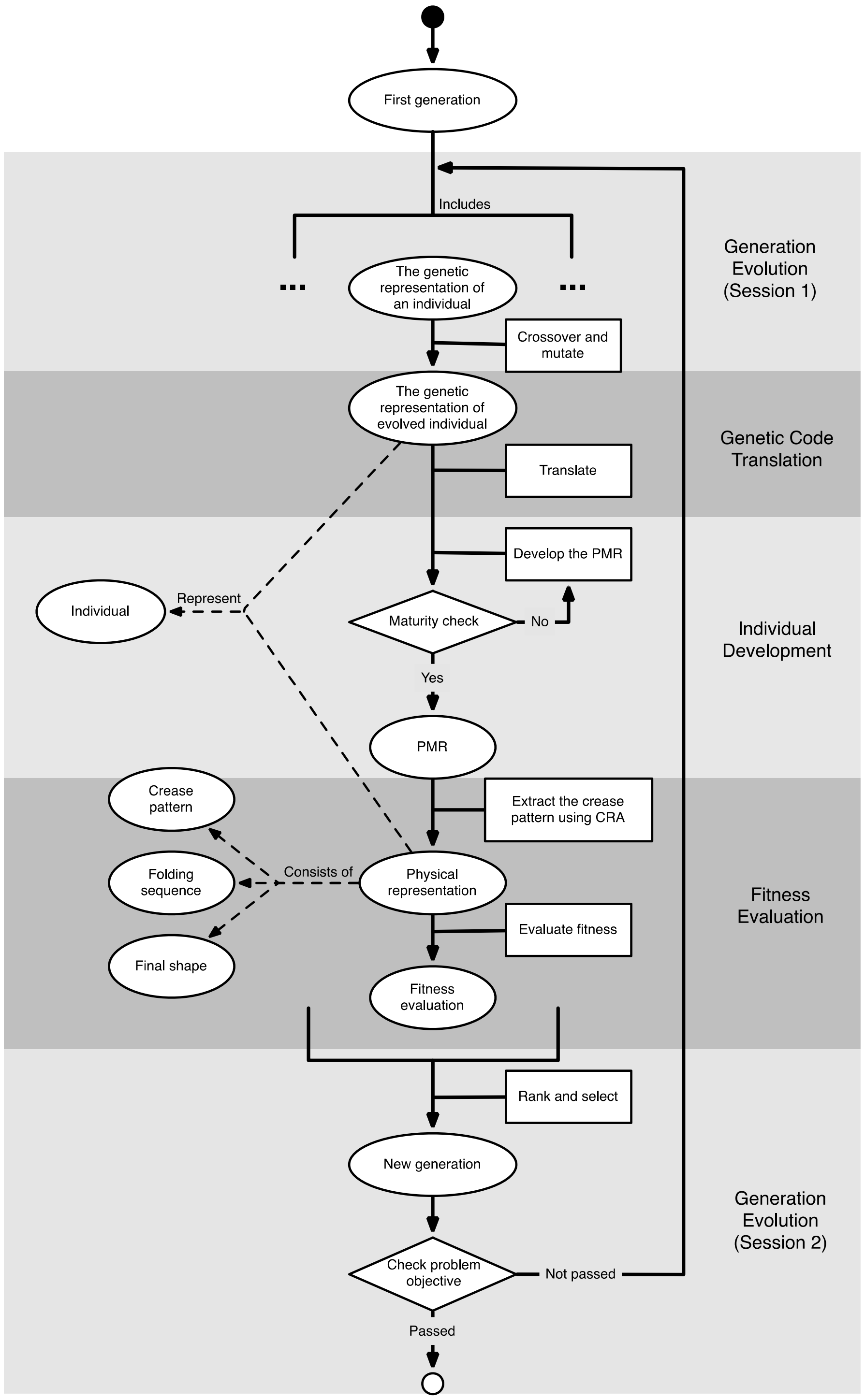


Figure 18 Workflow diagram of GA

The GA requires an initial generation of candidate solutions to start the evolutionary search. Then as in Figure 17, the method is recurrently run through the four main stages, which are “genetic code translation”, “individual development”, “fitness evaluation”, and “generation evolution”.

The “genetic code translation” stage implicitly converts the Gray code genetic representations of the candidate solutions into the descriptions or instructions for generating their physical representations. And in the “individual development” stage, the physical representations of the candidate solutions are explicitly formed and recorded by the GA.

After the first two stages, all the individuals in the current generation are physically realized. In the “fitness evaluation” stage, the GA assesses how well the physical representation, as well as genetic representation in several cases, of each candidate solution meets the design objective and constraints. Then the GA will derive a set of fitness values for the current generation according to the assessment.

Finally, in the “generation evolution” stage, the GA applies the evolutionary operators of crossover, mutation, and selection that are used to obtain a new generation from the current generation. The new generation will consist of the same number of individuals as the current generation. Then, the current epoch ends, and the next epoch starts. The evolution will be terminated as soon as we get the desirable solution or when the fitness value of the best individual in the generation converges.

Interpreting the Genetic Representation

For GA, the genetic representation of a candidate solution is a piece of genetic code in Gray code. The genetic code is usually the combination of segments, each of which represents an argument. Therefore the full genetic code can be understood as equivalent to a set of arguments that uniquely define the corresponding physical representation.

The format of partitioning the genetic code into segments, the way of converting each segment into the argument value, as well as the approach of using the arguments to describe the physical representation consists of the three key stages of interpreting the genetic representation. For each specific application of GA, there is always an implicit interpreter that manages the three stages to achieve the mapping between the genetic representation and the physical representation of each candidate solution.

For the “ice-cracking” representation, the mapping between the genetic representation and the physical representation has been clearly explained through Table 1. “Ice-cracking” implements the analogous procedure of ice cracks to develop the crease pattern that the genetic representation encodes. The encoded arguments in “ice-cracking” directly describe the vertices and creases, though in an unconventional way.

The PMR has the genetic representation to actually encode the arguments that define its cells. And it also requires an additional CRA to extract the implicit information of crease pattern. For this representation, the interpretation from the genetic representation to the physical representation is indirect through the intermediate PMR, which is defined as an equivalent representation of the crease pattern that it expresses.

Compared to “ice-cracking”, the PMR can also adopt some features from generative design. The genetic code has a hybrid structure that is the concatenation of two parts. The first part is an explicit code that encodes the *embryonic state* - the initial state of a PMR, and the second part is an implicit code that encrypts a set of generative genetic rules that guide cells to change in size or color accordingly, so as to make the entire embryonic PMR develope to its *mature state*. Since the PMR was inspired by the existing research on the CEE, the genetic algorithm combining the PMR is named computational evolutionary embryogeny for optimal origami design (CEEFOOD) in the rest of this paper.

The generative genetic rules adopt the same “if-conditionals-then-actions” format from CEE. In this study, three rules are designed as shown in Table 2, where N_b is the number of neighboring blue cells, N_r is the number of neighboring red cells, N_t is an integer argument defined in the rule, S_b is the total size of neighboring blue cells, and S_r is the total size of neighboring red cells, S_t is a real number defined in the rule. Therefore, two arguments defining each rule are the rule’s type number (1~3), and N_t or S_t for the conditional. According to Table 2, the essence of the three types of rules is to assimilate a cell’s color with its surroundings. Thus with these rules, a genetic code has a lower chance of resulting in an extremely random distribution of blue and red cell in the PMR.

Table 2 Generative genetic rules for PMR's genetic representation

Type #	Conditional	Action
1	$N = max(N_b, N_r) > N_t$	Convert the cell color to blue, if N_b>N_r; to red, otherwise
2	$S = max(S_b, S_r) > S_t$	Convert the cell color to blue, if S_b>S_r; to red, otherwise
3	$D = abs(S_b - S_r) > S_t$	Convert the cell color to blue, if S_b>S_r; to red, otherwise

The encoding of the genetic codes is very intuitive. For the explicit code part, we sequentially list all the arguments that define the cells in the embryo of a PMR and convert the arguments into Gray code. For the implicit code part, we extract the arguments (rule type #, N_t, and S_t) that define the generative genetic rules and convert the arguments into Gray code. The implicit code part also has several additional bits – called *rule switches*, each of which controls whether one of rules is active or not. If the rule switch bit of a rule is 1, the rule will be executed; if the rule switch bit is 0, the rule will be ignored in the individual development stage.

Individual Development According to the Generative Genetic Rules

Based on the embryo state and the generative genetic rules, the PMR of each individual will start developing from its embryonic state and update the sizes and colors of cells periodically. In each development period, every cell in the individual will go through all the generative genetic rules. If the present size or color of any cell satisfies the conditionals in a piece of generative genetic rule, the cell will execute the corresponding actions. For example, if a red cell is about to execute a rule - saying "if there are 2 more blue cells around than red cells, convert the cell to blue", we need to inspect the adjacent cells of this red cell first. If we find that the adjacent cells include 6 blue cells and 2 red cells, which satisfy the conditional of the rule, we take the corresponding action to recolor the red cell to blue.

A maturity check is set up as a criterion to decide when an individual's development stage terminates. Normally the development terminates if none of the cells will change in its size or color. The maturity check also sets a limit for the maximum development cycle. Any individual that doesn't stabilize within the limited number of cycles is terminated and labeled as "abnormal".

Here is an explanatory example of how a genetic rule leads a PMR to develop from its embryonic state to its mature state. The embryonic PMR defined by the explicit part of the hybrid genetic code is given by Figure 19(a). The individual has one generative genetic rule defined by the implicit part of the hybrid genetic code. The rule says "if a cell has more than 2 adjacent blue/red cells, convert the cell to blue/red". By definition, only the cells right above, right below, to the left of or to the right of a cell are its adjacent cells. So in the embryonic state of PMR, only the two cells highlighted by with green radiance satisfy the conditional of the genetic rule. Thus the two cells will execute the corresponding actions – the red cell in the middle converts to blue for having more than 2 adjacent blue cells, and the bottom middle blue cell converts to red for having more than 2 adjacent red cells. After the color change of the two cells, the PMR develops into a new state as shown in Figure 19(b). The maturity check will compare the two consecutive states in Figure 19(a) and (b). Finding that Figure 19(a) and (b) are not identical, the maturity check fails and the individual development will continue. So the state in Figure 19(b) is not the mature state, and we name it as an intermediate state. Then the next cycle of individual development starts with the intermediate state Figure 19(b) and run the rule on all the cells again. This time, the top middle cell is the only one that satisfies the conditional of the rule. Since this cell has more than 2 adjacent blue cells, its color will be assigned to blue, though the color doesn't change in fact. After deriving another new state as shown in Figure 19(c), the maturity check compares the states of Figure 19(b) and (c). Finding that the two states of the PMR are identical, the maturity check passes. Therefore, we mark the state Figure 19(c) as the mature state and terminate the individual development stage.

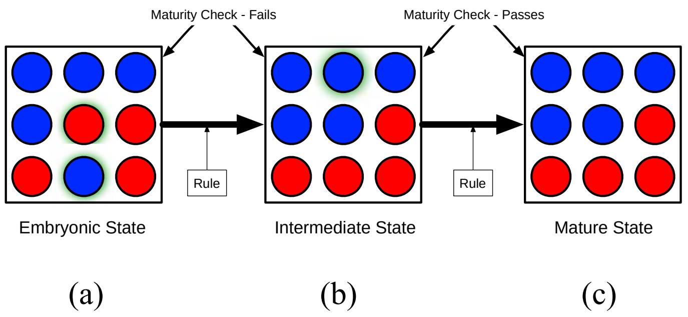


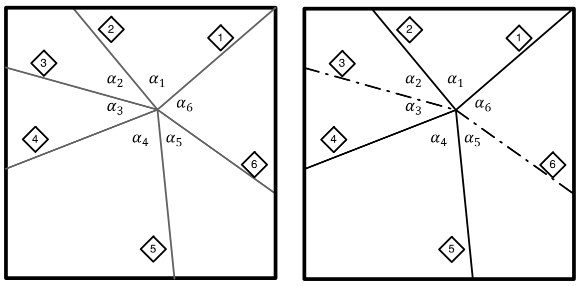
Figure 19 The individual development of a PMR with 3 by 3 cells. (a) the embryonic state of the PMR has 5 blue cells and 4 red cells; (b) the rule directs two cells to change their colors and guilds the PMR to an intermediate state; (c) The rules directs the PMR to develop into the mature state, which also has 5 blue cells and 4 red cells.

Evaluate the Fitness With Respect to Requirements

The automatic evaluation of every candidate solution is another essential component of GA. The definition of the objective function has been presented in the first chapter with the problem statement. In this section, the algorithms for deriving the fitness/objective components will be discussed.

Algorithm for Locally Flat-foldable Mountain-Valley Assignment

The validation of the flat-foldability of an origami crease pattern is equivalent to checking the existence of a mountain-valley assignment (MV-assignment) of the creases, according to which the faces won't collide with or pierce through each other as the shape is folded flat. Since every crease can be either a mountain of a valley, for a given crease arrangement with m creases, there are at most 2^m possible MV-assignments. For instance, the single vertex crease pattern with 6 creases (Figure 20) will have 2^6 MV-assignments theoretically. However, as we are looking for flat-foldable MV-assignments, some of the 2^6 MV-assignments are not qualified. The simplest non-flat-foldable case is when the 6 creases are all mountains, and all faces must be bent.



(a) (b)

Figure 20 (a) An example crease pattern with 6 creases. Creases are indexed by numbers in diamonds. The angles α_2 and α_5 are local min, thus creases #2 and #3 must be one mountain and one valley, and so do creases #5 and #6; (b) This displays one possible flat-foldable MV-assignment for (a).

Suppose that for each interior vertex, the angles among its linked creases are already designed to satisfy Kawasaki's theorem, so that the local flat-foldability will only depend on the satisfaction of Maekawa's theorem and the local min theorem. Unlike Kawasaki's theorem that is concerned about the angles of creases, Maekawa's theorem and the local min theorem define the requirements of the MV-assignment. Let's recall what the two theorems say. Maekawa's theorem asks that the difference of the numbers of mountains and valleys around a vertex must be 2. Therefore, for the same crease pattern in Figure 20, there are either 4 mountains and 2 valleys, or 2 mountains and 4 valleys. As a result, the maximal possible number of MV-assignments due to the request of flat-foldability will drastically decrease to

$$C_6^4 + C_6^2 = 2 \times C_6^2 \tag{27}$$

Then the local min theorem will further decrease this limit. For the crease pattern Figure 20, there are two local minimal angles, so the crease #2 must have the opposite MV with the crease #3, and the same for the pair of creases #5 and #6. In this case, the flat-foldable MV-assignment choices need to exclude the cases, in which any of the local min crease pairs have the same MV. It would be much easier if we separately consider those local min crease pairs from the very beginning. In general, a single vertex crease pattern has m creases, where $m = 2 \times M + 2 (M = 1, 2, 3, \dots)$, and N local min crease pairs, where $1 \leq N \leq M$. Each of the N pairs has two possible MV-assignments. And among the other $2 \times (M - N) + 2$ creases, there are either $M - N + 2$ mountains and $M - N$ valleys, or $M - N + 2$ valleys and $M - N$ mountains. The total number of flat-foldable MV-assignments will be

$$U_V(M, N) = 2^N \times \left(C_{2\times(M-N)+2}^{M-N+2} + C_{2\times(M-N)+2}^{M-N}\right) = 2^{N+1} \times C_{2\times(M-N)+2}^{M-N} \tag{28}$$

The value above can still become very large as M and/or N increase. Adding more vertices will make the situation more complex. Given a crease pattern has l interior vertices, we can derive the upper limit of the number of local flat-foldable MV-assignment for the i-th vertex is $U_{Vi}(M_i, N_i)$. And if there are m_c creases that are the ones connecting two vertices, the theoretical maximal number of possible flat-foldable NV-assignments of the full crease pattern will be

$$U_{CP}(M, N, m, m_c) = \left(\prod_{i=1}^{m} U_{Vi}(M_i, N_i)\right)/2^{m_c} \tag{29}$$

For a crease pattern, we can use the following algorithm to derive the full set of flat-foldable MV-assignments.

Algorithm: Deriving all the locally flat-foldable MV-assignments:

1 From a given crease arrangement, keep vertices indexed by numbers of 1 to l, and creases by 1 to m.

```
2     Create a library of existing MV-assignments – $L_{MV}$
3     Create a set of Boolean variables $Y_j = 0$ (j = 1, 2, m) to mark whether the
      algorithm has already assigned the direction for each crease.
4     For i = 1 to l
      4.1   Create an empty temporary library of MV-assignments – $\overline{L}_{MV}$
      4.2   For each existing MV-assignment $a_{MV}$ in $L_{MV}$
            4.2.1.1 Inspect the creases around vertex #i that has already been given
                    a MV in $a_{MV}$;
            4.2.1.2 Find out all the possible MV-assignments for the creases
                    around vertex #i that $Y_j = 0$, and store theses MV-assignments
                    in $\overline{L}_{MV}$
      4.3   $L_{MV} = \overline{L}_{MV}$
      4.4   For each crease #j that is connected with vertex #i
         4.4.1 $Y_j = 1$.
```

L_{MV} contains all the locally flat-foldable MV-assignments.Since the size of the solution is large, the computational complexity cannot be reduced. At this stage, the computer isn't able to make the judgment about whether a crease pattern is flat-foldable without trying each possible MV-assignment until reaching a flat-foldable one.

Algorithm for Global Flat-foldability Inspection

In the last section, I have given a discussion about flat-foldable MV-assignments. But the resultant flat-foldable MV-assignment can only guarantee the local flat-foldability everywhere. To inspect the global flat-foldability, it is necessary to exclude the MV-assignments that will cause face collision as all the creases are folded to $\pm\pi$. Therefore, the question becomes - having a crease pattern and a locally flat-foldable MV-assignment, how to check for the existence of face collisions.

Now, let's start with the simplest case, where a crease pattern has two parallel valley creases Figure 21. If we fold the crease #1 first, and then the crease #2, the shape is nicely flat-foldable. But as we reverse the sequence of folding, and start form the crease #2, the folding motion route of the crease #1 will be blocked by the folded face #3. For this example, we state that the crease pattern is globally flat-foldable, because the folding sequence given in Figure 21(b) folds the shape flat without causing face collision. But it is also true that not all the folding sequences for this globally flat-foldable shape can fold the shape flat.

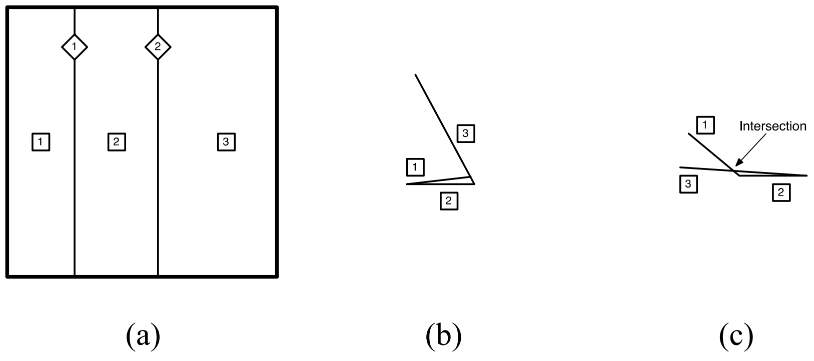


Figure 21 A crease pattern with two parallel valley creases and its folding sequences. (a) The crease pattern with indexed creases and faces; (b) Sequence 1 - crease #1 → #2: flat-foldable; (c) Sequence 2 - crease #2 → #1: not flat-foldable due to face intersection (collision, or pierce).

Now we proceed to a more complex example as shown in Figure 22(a), which has interior vertices. The pinwheel base has 12 creases, which include 4 mountains (dash-dot lines) and 8 valleys (solid lines). The MV-assignment presented in Figure 22(a) is locally flat-foldable everywhere. Figure 22(b) shows what the flat-folded shape looks like over the top, and it also proves that the pinwheel base is globally flat-foldable.

Let's think about how to prove global flat-foldability of this pinwheel base crease pattern. Recall the crease pattern in Figure 21. It has only simple folds that don't have intersections with each other. For a simple fold in a folding sequence, it is valid or flat-foldable, if and only if at the current state, all the faces have no intersection with the crease from the top view. In Figure 22(a), the creases of the pinwheel base crease pattern are all linked through interior vertices, which means these linked creases must be folded simultaneously. Thus in this case, we cannot use the same approach for Figure 21.

In theory, the flat-foldability only talks about the static properties of the shape when all the creases are folded to $\pm\pi$, regardless of how difficult the shape folds physically, or how many auxiliary folds the user has to make during the procedure to reach the final shape. Therefore, we will look for an approach that doesn't track the folding sequence, but only investigates whether the faces can be arranged in a certain order from bottom to top without causing intersections.

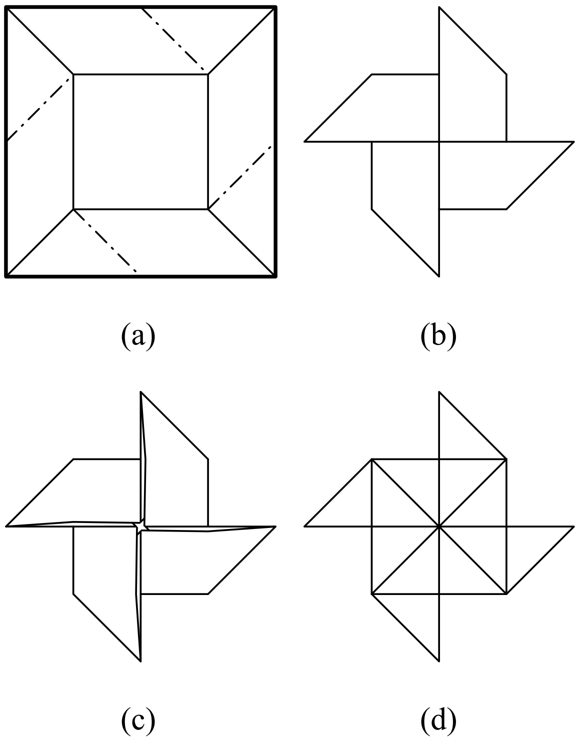
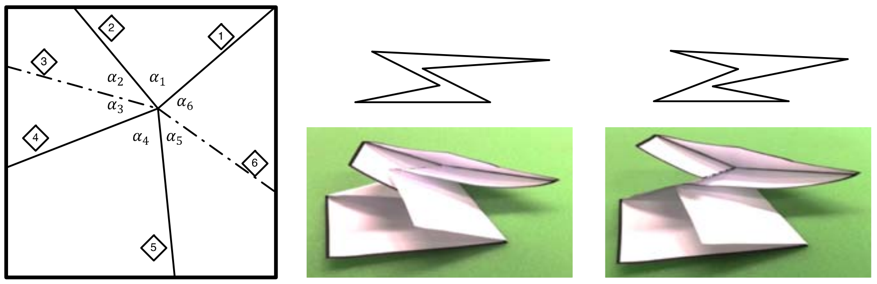


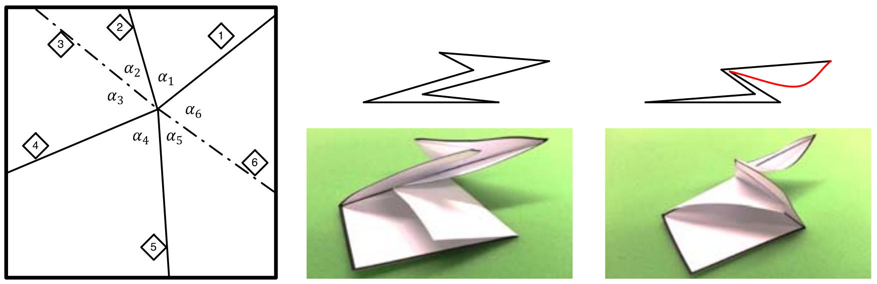
Figure 22 A pinwheel base and its flat-folded state.

For the crease pattern shown in Figure 20(b), we can fold it flat in two ways. Using the face between creases #4 and #5 as the bottom one, we can either fold the flap of crease #3 below the flap of crease #6 (Figure 23(b)), or fold the flap of crease #6 below the flap of crease #3 (Figure 23(c)). To testify the global flat-foldability of the crease pattern, we need to at least seek out one of the two flat-folding ways. As an extensive study, we rotate creases #2 and #3 in Figure 23(a) clockwise to derive the crease pattern in Figure 24(a), so that α_1 becomes smaller and α_3 larger. Folding the altered crease pattern flat, we can only place the flap of crease #3 above the flap of crease #6 Figure 24(b). If we try to push the flap of crease #6 into the limited space between the two faces adjacent to crease #2 (Figure 24(c)), the face between creases #1 and #6 will have to bend to fit in. However, the crease pattern in Figure 24(a) is still globally flat-foldable, since it has at least one legal way to fold flat.



(a) (b) (c)

Figure 23 A single vertex crease pattern and two ways of flat-folding it.



(a) (b) (c)

Figure 24 An altered crease pattern from Figure 12. It has only one way to be flat-folded as in (b). Using the way shown in (c) would cause the face highlighted to bend within the limited space closed by the two faces adjacent to crease #2.

From the last two examples, especially the one of Figure 24, we see that the order of faces overlapping with each other is crucial for determining whether the way of folding results in a flat-folded shape. Therefore, finding out the way of flat folding a shape is equivalent to finding out a face overlapping order, according to which none of the faces will have to crumple or bend, while the creases are folded to $\pm\pi$.

The most straightforward approach to detect the existence of a flat-foldable face overlapping order is to go through every possibility until we find one out. This brute force method is computationally complex. But since the problem of validating a crease pattern flat-foldable has been proved NP-complete, a brute force method doesn't sound worse.

But for such a brute force search, the difficult part is in how to verify if a face overlapping order directs the faces to fold flat. My solution is to assume that the faces have zero thickness and collision size. So the faces can fold freely in the space and piercing is also allowed. Then after calculating the ultimate position and orientation of each face at the flat-semifolded state, we check whether there exist some instances of face piercing according to each face overlapping order. The following criterion provides the sufficient condition for the existence of face penetration. Criterion: At the flat-semifolded state, if there is a crease #j partially or fully falling inside the region of a face #k, then the face #k cannot be placed between the two faces adjacent to crease #j in the face overlapping order. Otherwise, face #k will pierce through crease #j.

The computational complexity of applying this criterion is $O(mnp)$, where m is the number of creases, n is the number of faces, and p is the average number of creases that each face has. And the application of this criterion will return a table T_{FP} with m rows and n columns. The contents are Boolean values telling whether a face can be placed between the two flaps of a crease. After being fully filled, this table can be repetitively referred to by every candidate face overlapping order.

Then go up one level to review the brute force search of all possible face overlapping orders. The ordering of n faces should be $O(n!)$. But if we also put the MV-assignment into consideration, we will find that from the standpoint of any arbitrary face, its adjacent faces that are linked by mountain creases can only be folded below, and its adjacent faces that are linked by valley creases can only be folded above. Thus, the face overlapping orders that break this pattern should be ruled out. As a result, the brute force search can be simplified to a tree-search featuring branch-trimming. The algorithm of the tree search can be described as:

Algorithm: Tree search of face overlapping orders (with respect to a crease pattern and a given MV-assignment)

```
1   Select the face #$k_r$ to be the root that is fixed. The root face is also expressed as
    $f_r$
2   Create a queue of faces $Q_F$ and a queue of creases $Q_c$
    2.1   For each crease $c_i^r$ that is on the boundary of $f_r$
          2.1.1 Push $c_i^r$ into the back of $Q_c$
          2.1.2 Push $f_i^r$ into the back of $Q_f$, where $f_i^r$ is the face that is linked with
                $f_r$ over the crease $c_i^r$
3   Create an empty library of creases $L_C$
4   Create a library of currently found face overlapping orders. The library is
    named $L_{FO}$, and the i-th face overlapping order in this library is expressed as $o_i$.
    4.1  Initialize the library $L_{FO} = \{o_1\}$ with only one member of face
         overlapping order $o_1 = \{f_r\}$.
    4.2  A face overlapping order variable $o_i$ is a sequence of faces, where the
         order of the faces in the sequence is the order of the faces in the
         corresponding face overlapping order from bottom to top.
5   Create and initialize a Boolean vector $B_{FO}$ of $n-1$ “false”s, and one “true” at
    the $k_r$-th element
6   While $Q_c$ is not empty
    6.1  In this iteration, we look for all the possible placements of the first
         element $f_1^Q$ in the queue $Q_f$ in each currently found face overlapping
         order in the library $L_{FO}$.
    6.2  Create an empty temporary library of face overlapping orders $\overline{L}_{FO}$
    6.3  For each $o_i$ in $L_{FO}$
        6.3.1 For each place that we can insert $f_1^Q$ in $o_i$, including the beginning,
              the end or the position between any two faces in the sequence, if the
              placement isn’t in conflict with the criterion expressed by the table
              $T_{FP}$, then push the new face overlapping order $\overline{o}_i$ ($o_i$ with $f_1^Q$ inserted)
              to the back of $\overline{L}_{FO}$
    6.4  $L_{FO} = \overline{L}_{FO}$
    6.5  For each crease $c_i^{Q1}$ that is on the boundary of $f_1^Q$ but not a member of $L_C$
        6.5.1 Push $c_i^{Q1}$ into the back of $Q_c$
```

6.6 Push f_i^{Q1} into the back of Q_f, where f_i^{Q1} is the face that is linked with f_1^Q over the crease c_i^{Q1}

6.7 Pop the f_1^Q out of Q_f

6.8 Pop the first element c_1^Q of Q_c out, and push it to the back of L_C

7 $\mathrm{L_{FO}}$ stores all the globally flat-foldable face overlapping orders. And if $\mathrm{L_{FO}}$ is empty, the crease pattern with the corresponding MV-assignment is not globally flat-foldable.

The algorithm above gives a systematic approach to thoroughly seek out all the possible face overlapping orders of a crease pattern to be folded flat. The computational complexity of this algorithm depends on the number of mountains and valleys. For the best situation, the complexity would be expected to be $\mathrm{O}(n!/2^n)$.

Calculating the Flat-folded State Profile Area

The flat-folded state profile area is the key objective of the design problem given in the chapter I. However, it is not difficult as long as the design has been proved flat-foldable. And of course, if the shape is not flat-foldable, there is no need or no way to calculate the profile area.

The method of calculating the profile area has two steps:

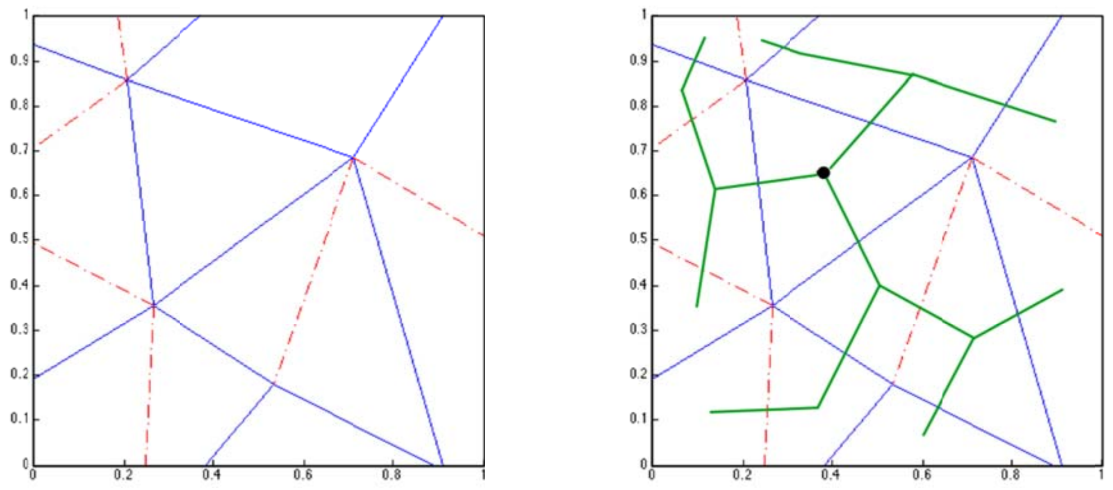
Step 1: Simulate the folding and calculate the orientation of the faces. A very basic method can be used for this step.

Step 1.1 – Fix one face onto a Cartesian CS.

Step 1.2 – Use the stationary face as the root to build a tree graph. In the tree graph, each child node is a face connected to its parent. And the depth of each child node representing face must be minimized in the tree graph. One of the possible tree graphs for faces in the crease pattern Figure 25(a) is given as Figure 25(b), where the root node (the stationary face) is marked by a black dot. In the tree graph, each edge represents the crease that connects its two nodes (two faces).

Step 1.3 – Flip each face several times along the creases represented by the edges, which form the path from this face up to the root face, sequentially. Record the orientations of the flipped faces.

Step 2: Compute the Boolean union of the flipped faces as well as the root face. The area of the union shape is the flat-folded state profile area.



(a) (b)

Figure 25 A crease pattern and tree structure of its faces

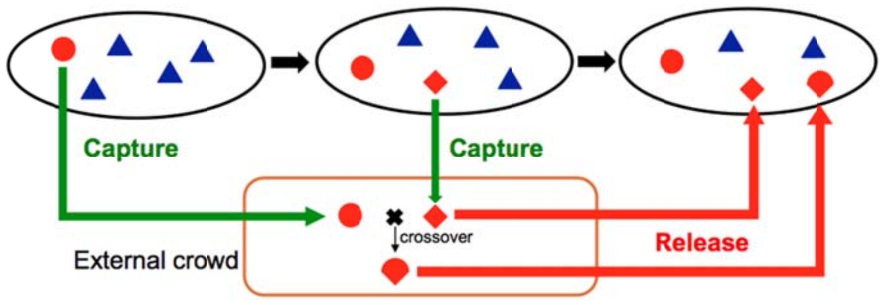
Design the Evolutionary Operators and Complementary Mechanisms

The GA is a search heuristic inspired by the process of natural evolution. It is generally used to derive an optimized solution through iteratively proposing, adapting, and selecting candidate solutions. The most noteworthy advantage of GA methods is that it avoids formulating an explicit model for the embedded design search space by constructing a substitutive search space defined by the genetic code. Thus GA methods generally provide an effective approach for a random search in both a well-defined design space as well as the design space that the designers do not intentionally seek.

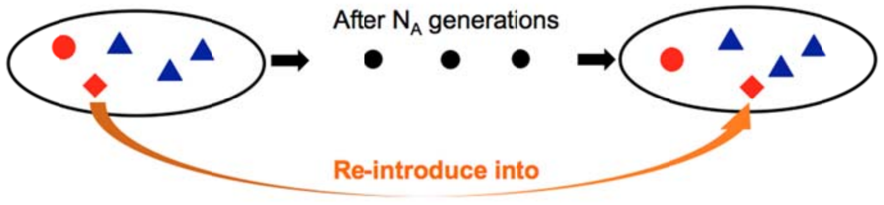
To derive optimal solutions, the GA methods must possess the capability of performing both broad exploration and deep exploitation. The broad exploration guarantees the diversity of the coming candidate solutions, while the deep exploitation preserves the once-emerged elite solutions. This balance between exploration and exploitation, which is also a balance between diversity and elitism, is the cardinal objective of making modifications on GA's basic evolutionary operators.

Operators for Elitism

In this research, the measures of elitism preservation are scheduled in accordance with two considerations: to prevent the elites' extinction and to restore the extinct elites. The extinction of the emerged elites is usually caused by faulty selection or unexpected mutation. The simplest but most reliable way to prevent the extinction of the elites is to ensure that the elites are always selected into the next generation or to diminish the probability of mutating elites. On the other hand, restoring the extinct elites requires some complementary mechanisms to make backups of candidate solutions. If the complementary mechanism backs up the elites in an external storage, it is called captive breeding; while if it backs up the elites in an "internal" storage, it is called atavism.



(a)



(b)

Figure 26 The illustration of captive breeding (a) and atavism (b).

In ***captive breeding*** (Figure 26(a)), an external crowd will be created to store the elite individuals once they emerge. This mechanism is developed based on the external Pareto archive used in (Jensenm 2003, Knowles and Corne 1999, Zitzler and Thiele 1999). Sometimes, we alternatively restrict the captive breeding storage size by only allowing non-dominated individuals. When a newly captured individual comes to the captive breeding storage, it is compared with all the current members in the crowd. While the external Pareto archives only store elites, the external storage of captive breeding allows crossover operations among its members. The children members have the chance to inherit the advantages from both their parents, and thus become the elites among elites. Thereon when the GA is executed in each generation, $\mathrm{n_b}(< \mathrm{N_b})$ randomly chosen members will be released from the captive breeding crowd into the evolving population to compete with the entire current generation of individuals.

Another complementary mechanism called ***atavism*** stores the evolving population (Figure 26(b)). Atavism functions by directly introducing the elite individuals from the generation of N_a epochs back into the current generation. Atavism essentially regulates the evolution direction from deterioration due to unwanted mutation of an elite individual.

Operators for Diversity

Although captive breeding and atavism work well to maintain the elite individuals in the population, they also cause severe pre-mature convergence toward the early-emerged elites. If the fitness evaluation is scalar, a higher mutation rate, the stochastic universal sampling (SUS) selection, and injection of random candidate solutions can be applied to increase the diversity within the population so as to neutralize the extreme elitism. A higher mutation rate, which extends the search step size, not only prevents the search from being trapped into a local optimum, but also enables a greater variety of solutions. Nevertheless, SUS selection permits some of the less fit candidates to be selected. Using SUS selection has the non-negligible risk of causing elites to be eliminated accidentally. If the captive breeding and atavism mechanisms are used, however, the lost elites are returned to the population. Injecting randomly generated candidate solutions into each generation to take part in the crossover and selection processes also broadens the solution search region.

Another diversity metric that will be considered in this research is the use of a multi-objective fitness evaluation, as the origami design or self-folding control problems usually involves more than one requirement. Under multi-objective GA, the fitness of each candidate solution is represented by a *fitness vector*, which is formed by the unified fitness evaluation components. The individuals are ranked through multi-objective Pareto ranking (Goldberg 1989, Konak, Coit and Smith 2006). A candidate solution from a higher frontier is always more preferable than any one from a lower frontier. The candidate solutions from the non-dominated Pareto frontier are the elites. Unlike a weighted single objective fitness ranking relying on a sort of "overall" performance, the multi-objective ranking and multi-objective GA is fundamentally open to more "specialized" elites rather than just the "all-around" elites found by a normalized fitness ranking, and thus increases the diversity among the elite class of the population. However, the non-dominated Pareto frontier will sometimes contain more candidate solutions than are wanted for selection. Under such circumstances, normalized fitness evaluation is applied to give a ranking inside each frontier.

During CEEFOOD evolution, in addition to the GA operators of selection, mutation and crossover, we also apply another special operator called code assimilation. In this research, we use the term *solution space* to name the domain of the physical representation; and use the term *search space* to name the domain of the genetic representation. As defined in prior sections, the physical representation is generally the crease pattern, but in this research it directly represents the PMR that is described by the genetic representation, and indirectly indicates the crease pattern that is derived by using the CRA on the PMR.

Here, we need to clarify that appending the generative genetic rules (the implicit code part) in the genetic representation of the candidate solutions doesn't expand the scope of the solution space. The setup of the cell sizes and colors in the PMR doesn't change, so the total number of possible PMRs doesn't change either. But the implicit code will increase the original design search space by creating multiple instances for each possible solution. These instances share the same physical representation, but different genetic representation. They are different genetic code and rule routes to get to the same candidate solution.

Among the instances of one physical representation, there is always at least one, whose explicit genetic code part defines the embryonic PMR having the exact identical physical representation as the mature PMR of the other instances, while its implicit code part has all the rule switch bits to be 0. We name these instances with their genetic rules switched off the *non-aging instances*. With all the rules turned off, the embryonic PMR will remain unchanged during the individual development stage, so its mature state will be exactly the same. For example, consider a simplified setting of PMR, which has only four cells with only color being customizable. In the hybrid encoding, the explicit part has four bits, the first defines the color of the upper-left cell, the second the upper-right cell, the third the lower-left cell, and the fourth the lower-right cell. A bit of 0 indicates blue and 1 indicates red. For the four candidate solutions, whose individual development procedures are shown in Figure 27, their rules and rule switch states are all listed. Though the explicit code parts of the four candidate solutions define different embryonic PMRs, their rules guild them to develop into the same mature PMR. Since the candidate solutions in Figure 27(c) and (d) have their rules switched off, they are the non-aging instances of Figure 27(a) and (b).

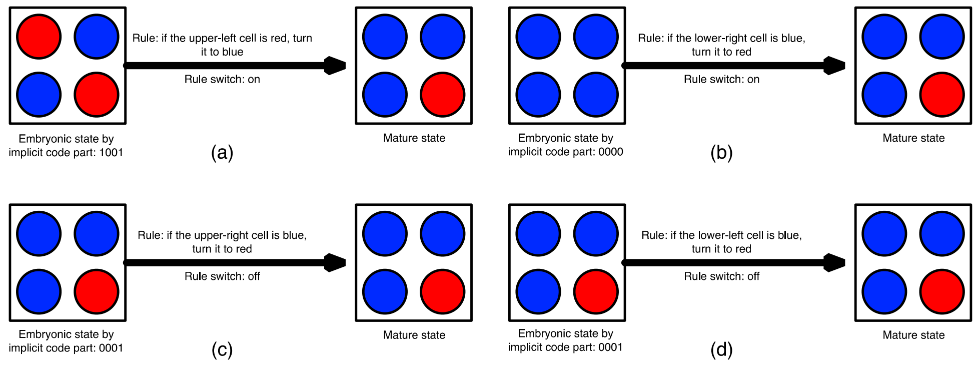

Rule: if the upper-left cell is red, turn it to blue
Rule switch: on
Embryonic state by implicit code part: 1001
(a)
Mature state
Rule: if the lower-right cell is blue, turn it to red
Rule switch: on
Embryonic state by implicit code part: 0000
(b)
Mature state
Rule: if the upper-right cell is blue, turn it to red
Rule switch: off
Embryonic state by implicit code part: 0001
(c)
Mature state
Rule: if the lower-left cell is blue, turn it to red
Rule switch: off
Embryonic state by implicit code part: 0001
(d)
Mature state

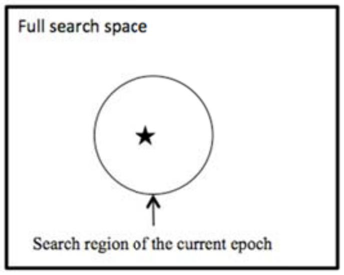
Figure 27 The individual development of four equivalent instances. For having a switch-off rule, (c) and (d) are both non-aging instances.

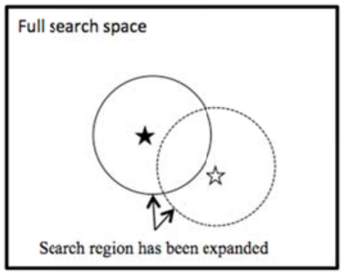
Code assimilation is an operator that introduces the non-aging instance equivalents of some existing candidate solutions into the current generation. If our PMR setting is as that of the candidate solutions in Figure 27, and we have instance Figure 27(a) in the current generation, code assimilation is to introduce either or both of Figure 27(c) and (d).

Code assimilation accelerates the evolution by expanding the search region based on the existing individuals in each epoch. And because the appendage of the implicit code part enlarges the search space and lowers the solution search efficiency, the code assimilation is necessary for neutralizing these drawbacks. For instance, suppose that the big box in Figure 28(a) defines the entire search space, and the “solid star” shows the location of a candidate solution. The solid circle is the current search region, where the evolved solutions - the products of the “solid star” solution through evolutionary operators – will locate. With the code assimilation, the non-aging equivalent of the “solid star” solution is introduced in. Its location in the search space is shown by a “hollow star” in Figure 28(b). The dashed circle represents the region, where the evolved solution based on the “hollow star” will locate. After applying code assimilation, new candidate solutions will be found in the union search region of the two circles instead only in the solid circle.

Elitism and Diversity

Elitism and diversity seem to be two contrasted concepts just like the white and black color shown in Figure 29. But in fact, the measures proposed for strengthening the elitism and for expanding the diversity need to cooperate in the frame of GA and CEE very well, because they are designed to utilize the elitism and diversity as the centripetal forces that prevent each other from escaping to extremity. For instance, the elites that unexpectedly got lost during pursuing diversity among solutions will eventually return through captive breeding and atavism; while the pre-mature converge caused by elitism will also be moderated by the multi-objective ranking.





(a) (b)

Figure 28 The code assimilation expands the search region

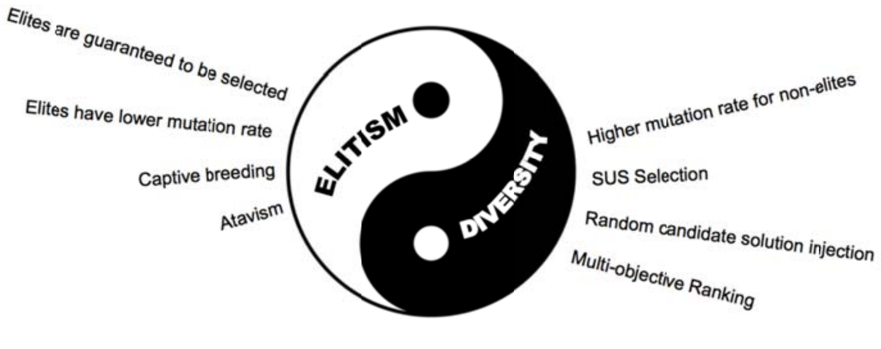

Elites are guaranteed to be selected
Elites have lower mutation rate
Captive breeding
Atavism
ELITISM
DIVERSITY
Higher mutation rate for non-elites
SUS Selection
Random candidate solution injection
Multi-objective Ranking

Figure 29 The synthesis among the measures that strengthen the elitism and the ones that increase the diversity.

CHAPTER V

ORIGAMI DESIGN RESULTS AND DISCUSSION

In this chapter, the results from the two prior chapters will be applied to the open origami design problem stated in the first chapter. I will present the designs that are derived by the genetic algorithm with the direct “ice-cracking” representation and the genetic algorithm with indirect PMR respectively. As the PMR was inspired by the existing research on the CEE, so the genetic algorithm combining the PMR is also an application of CEE, and it is, therefore, named computational evolutionary embryogeny for optimal origami design (CEEFOOD).

Then the designs, as well as the procedure of how the method arriving at the designs, will be put into comparison. This chapter will make discussions about these designs, so as to display the advantages of the abductive design method for open origami design problems.

Design Results by Genetic Algorithm with “Ice-cracking”

In this application, the GA applies the *hybrid ranking* approach, which includes the multi-objective ranking for separating candidate solutions into tiers and the normalized objective ranking for assessing the quality of the candidate solutions within the same tier. In hybrid ranking, the ones from a higher tier are always treated as more favorable solutions than those from a lower tier, no matter what their normalized fitness values are.

The desired flat-folded shape profile area A_t is set to 0.25. Since the "ice-cracking" representation implicates the guarantee of flat-foldability, thus the $Err_{flat-foldability}$ – that is required for calculating the term $\widehat{H_f}$ in the original objective function – is always 0, I add another two terms for this application, which request the system DOF to be lower than 5 and the shortest crease to be longer than 0.2. So the normalized fitness function (objective function) is

$$F = w_a \overline{H_a} + w_e \overline{H_e} + w_{dof} \overline{H_{dof}} + w_c \overline{H_c} \tag{31}$$

where

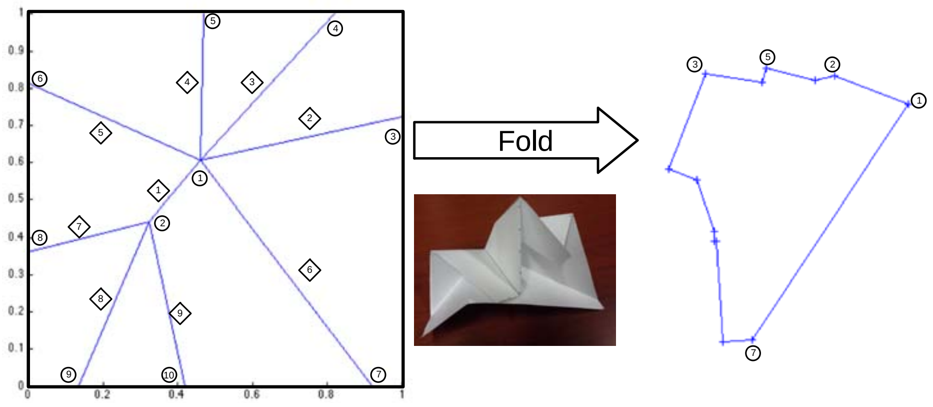
$$\begin{cases} \overline{H_a} = \left|A_{ff} - 0.25\right| \\ \overline{H_e} = \left|E_p\right| \\ \overline{H_{dof}} = \begin{cases} 1, if\ n_{DOF} > 6 \\ \dfrac{n_{DOF} - 1}{5}, if\ n_{DOF} \leq 6 \end{cases} \\ \overline{H_c} = \begin{cases} 0, if\ L_{minc} > 0.2 \\ \dfrac{0.2 - L_{minc}}{0.2}, if\ L_{minc} \leq 0.2 \end{cases} \end{cases}$$

A_{ff} is the flat-folded state profile area, E_p is the displacement of the CoM, and L_{minc} is the length of the shortest crease. The penalty term $\hat{P}(m)$ is 0 when the number of creases m is in the range of $\{M_L, M_U\} = \{1, 25\}$; otherwise, $\hat{P}(m)$ is 1. The weight vector is assigned to $\{w_a, w_e, w_{dof}, w_c\} = \{0.28, 0.41, 0.1, 0.21\}$ in this application.

Then for the multi-objective ranking, the fitness vector is $\vec{\mathrm{F}} = \{\mathrm{f_1}, \mathrm{f_2}, \mathrm{f_3}, \mathrm{f_4}\}$, where

$$\begin{cases} f_1 = \overline{H_a} \\ f_2 = \overline{H_e} \\ f_3 = \overline{H_{dof}} \\ f_4 = \overline{H_c} \end{cases}$$

Using the hybrid fitness ranking with the four above components, the GA derives a design with 2 vertices, 9 creases and 8 faces as in Figure 30. This design fits the problem objective and constraints very closely. Due to the fact that the design has A_{ff} of 0.25013353, E_p of 9.6957×10^{-5}, DOF of 2, and L_{minc} of 0.2119, its fitness vector is $\{1.3353 \times 10^{-4}, 9.6957 \times 10^{-5}, 0.2, 0.0595\}$.



(a) (b) (c)

Figure 30 (a) A crease pattern design (normalized fitness of 0.0208) found through the GA for the problem in Section 4, where vertices are labeled by numbers in circles, and creases by numbers in diamonds; (b) The intermediate folded state; (c) The corresponding flat-folded state profile that has an area of about 0.2501. Five vertices are still on the boundary of the profile.

The design shown in Figure 30 has excellent performance based on the fitness criteria of a flat-folded profile, change on center of mass, and the shortest crease. The number of DOF still has the potential to be further optimized to 1, as the current n_{DOF} is 2.

However, at this stage there are several open questions for this design problem that the GA could not yet give solutions to:

1) Is this design the global optimum, as there do exist other designs, such as Figure 31 that have very similar performances to the design shown in Figure 30 on all considered measures?
2) Before the genetic algorithm actually generates a solution as in Figure 32 with a nearly 0 fitness value, can we conclude whether a perfect design (a design with all its fitness vector components being 0) exists or not.

Nevertheless both the designs in Figure 30 and Figure 31 have already solved the problem approximately. Of course, the theoretically ultimate target is to get a design like Figure 32.

The essence of the NP-completeness of the open origami design problem determines that it is extremely difficult to precisely locate the global optima even by intelligent heuristics. Therefore, it can be stated that the three designs shown above are all acceptable and produce good results.

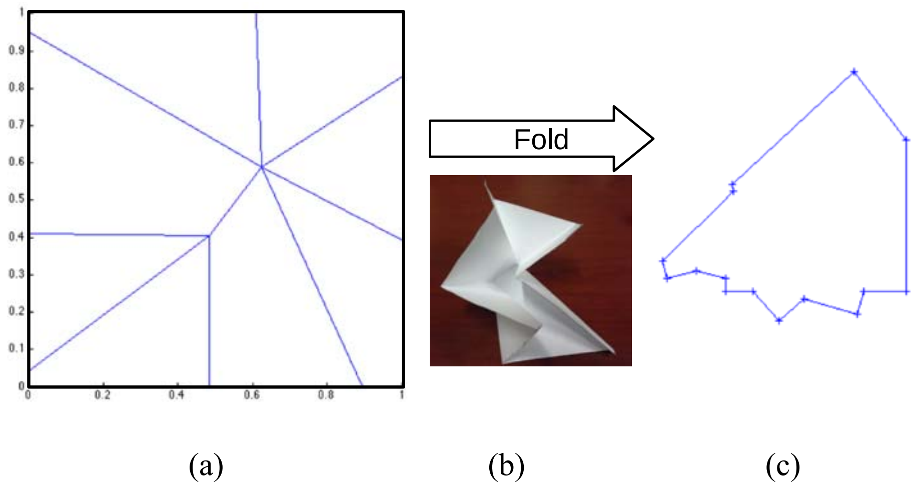


Figure 31 (a) A second crease pattern design (normalized fitness of 0.0209), which is very similar to Figure 30; (b) The intermediate folded status; (c) The corresponding flat-folded state profile that has an area of about 0.2499.

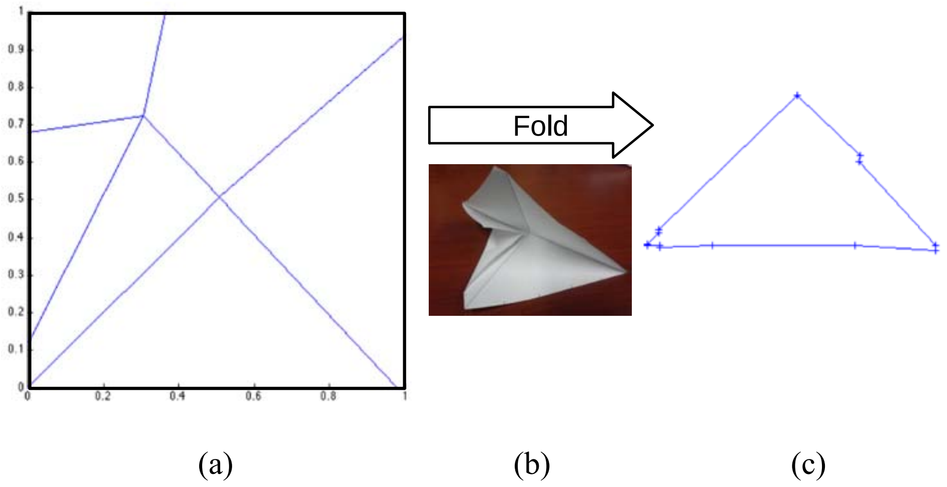


Figure 32 (a) A third crease pattern design (normalized fitness of 0.0035, and fitness vector of {0.0013, 0.0077, 0, 0}), which is almost an ideally optimal design; (b) The intermediate folded status; (c) The corresponding flat-folded state profile that has an area of about 0.2577.

Discussion

The plots below will entail how the three designs evolve over generations under the control of the GA. Figure 33, Figure 34 and Figure 35 are respectively for the designs Figure 30, Figure 31 and Figure 32. In the three plots, the blue solid lines show the normalized fitness value of the best candidate solution in each evolutionary generation. Such best candidate solution must be in the non-dominated tier (highest tier) according to the multi-objective ranking, and must be with the least normalized fitness value within its tier.

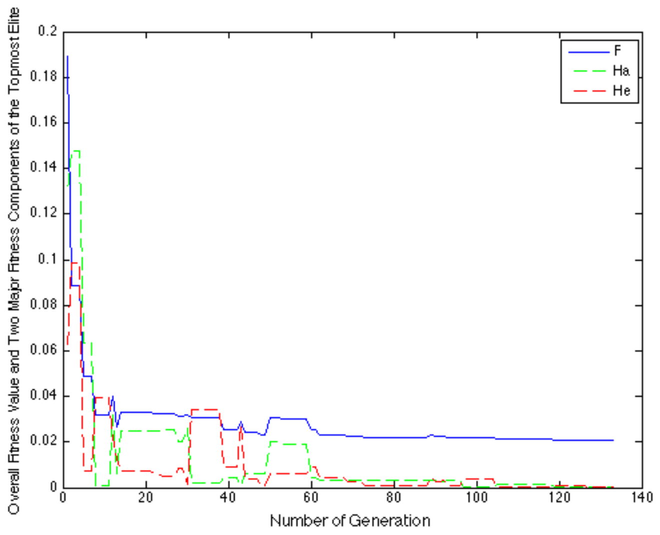


Figure 33 The fitness values of the topmost elites in the family of design Figure 30. *F* indicates the normalized fitness, the fitness component evaluating the flat-folded state profile area, and the fitness component evaluating the overlap of a crease with the center of mass.

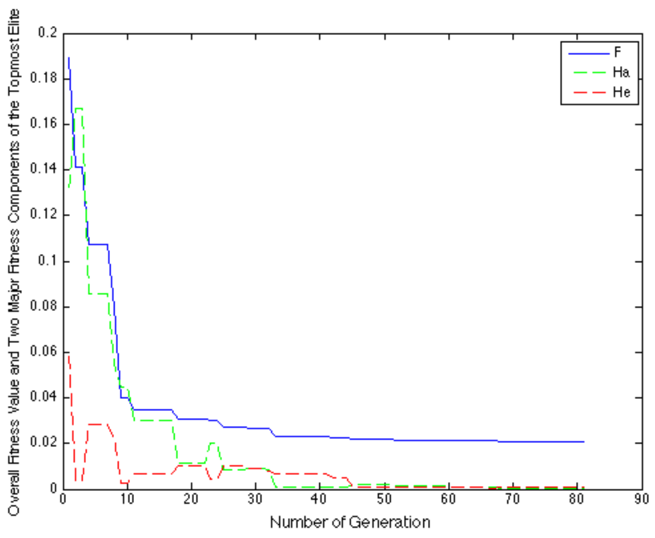


Figure 34 The fitness values of the topmost elites in the family of design Figure 31. *F* indicates the normalized fitness, the fitness component evaluating the flat-folded state profile area, and the fitness component evaluating the overlap of a crease with the center of mass.

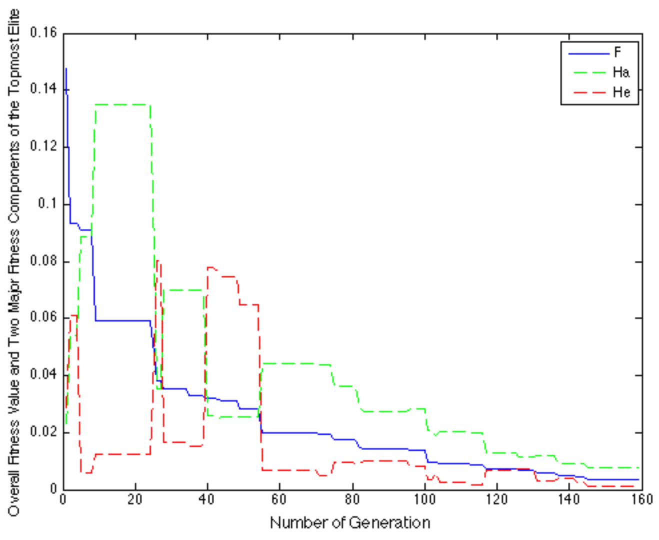


Figure 35 The fitness values of the topmost elites in the family of design Figure 32. *F* indicates the normalized fitness, the fitness component evaluating the flat-folded state profile area, and the fitness component evaluating the overlap of a crease with the center of mass.

For the plots, we can have a conclusion about one common feature of the curves. As the normalized fitness generally decreases constantly, $\overline{H_a}$ and $\overline{H_e}$ curves will have spikes upward or downward, especially during the early generations. This feature tells that the evolution of the GA starts with greater diversity, and will gradually converge to the optimal results. When there is greater diversity, the fitness components will have large differences among the candidate solutions in a generation. As the method proceeds, the non-dominate tier will shrink toward the region of the optima, thus the difference among candidate solutions will be less obvious. And the candidate solutions within the non-dominated tier will tend to develop into very close performance based on the four fitness components.

One particular thing in Figure 35 that is different from Figure 33 and Figure 34 is that though the normalized fitness of design Figure 32 is better than the other two, the $\overline{H_a}$ and $\overline{H_e}$ evaluations are in fact less optimized. This tells that the designs Figure 30 and Figure 31 both have $\overline{H_a}$ and $\overline{H_e}$ evaluations extremely close to the theoretical minimum of 0. But the two designs have n_{DOF} of 2, which greatly degrades their normalized fitness. If we review the normalized fitness function, we can find that one extra n_{DOF} is equivalent to about 0.07 of increase in $\overline{H_a}$. Considering a difference of 0.07 in $\overline{H_a}$ is significant for a desired flat-folded area of 0.25, the current normalized fitness functions implicates that the problem has a greater preference on looking for a design with $n_{DOF} = 1$.

However, the three designs can all be accepted as the solution of the problem. To make a decision on the one to take as the final solution depends on the designer's preference on the performance indices.

The GA with "ice-cracking" representation is a useful tool to help designers search for possibilities within the design space when they don't have full information. And the GA also accelerates the solution search for such a problem featuring NP-completeness and a nonlinear design space.

Design Results by Genetic Algorithm with PMR (CEEFOOD)

In this application, I will use the PMR of crease patterns with cells that are arranged in a variant square lattice (Li and McAdams 2013) as shown in Figure 14(a), which is derived by rotating a square lattice by 45°. The variant square lattice applied in the experiment will have a higher resolution so that the nearest distance among nodes is 0.0707. The whole square sheet region is defined by the desired shape $\boldsymbol{S}_t$. The size s_i of i-th cell varies from 0.01 to 0.08 with an interval of 0.01, so that $s_i = 0.01 \times n_i,\ n_i \in \{0,1,2,\cdots,8\}$. The cell color c_i is either blue or red ($c_i \in \{1,2 \mid 1\ means\ blue, 2\ means\ red\}$). The respective fitness evaluation formula (4.1) uses a weight vector of $\vec{w} = \{w_e, w_a, w_f\} = \{0.5,0.3,0.2\}$. The values of the weight factors are determined according to the preferred fitness function components ($\overline{H_e}$, $\overline{H_a}$, and $\overline{H_f}$). The component representing the feature given higher priority or preference will be assigned a larger weight.

Unlike the application of the GA with "ice-cracking", this application won't implement the hybrid ranking. But as the indirect PMR is more complicated in representing an origami, I focus on studying how some initial guesses can affect the solution search. Therefore, I separate the different trial runs of solution generation into two groups.

The first exercise group starts the evolution with an initial guess given manually and intuitively. The initial guess as shown in Figure 36(a) has 5 creases.

The initial guess doesn't have a genetic representation, thus one must be extracted to start the CEEFOOD process. Two methods can be used to get the genetic code for a given crease pattern. In the first method, we color the faces in the original initial guess by two colors as shown in Figure 36(b), and use the classical CEE to replicate this pattern in the same manner as the CEE generating colored maps (Bowers 2005). Then we pick the genetic code of the optimal replica as the genetic of the crease pattern. The second method will also require us to color the crease pattern as in Figure 36(b). In the next step, we manually design a PMR of the crease pattern based on the coloration. In this empirically determined PMR, each cell is assigned to the color of the face, in which this cell is located. The size of a cell can be randomly given, but if a cell's distance to its nearest crease is shorter than the maximum allowable size of the cells (0.08 according to the experiment setting), the size of this cell is set to the closest value (cell size can only be $0.01 \times n_i,\ n_i \in \{0,1,2,\cdots,8\}$) to the distance. The same setting of the PMR will be used by the second exercise group that will be introduced in the remainder of this section. Although the first method can generate a more precise replica of a given initial guess, the second method is much faster by avoiding the progressive searching process of the CEE. In this application, we apply the second method to obtain Figure 36(c), which is a close replica of Figure 36(a). And the Figure 36(c) will be an actual initial guess that is used by the CEEFOOD to start the evolutionary design. The crease pattern of Figure 36(c) has an overall fitness value F of 0.1795, with a flat-folded profile area of 0.3517 ($\overline{H_a}$=0.0739) and a potential energy evaluation $\overline{H_e}$ of 0.2147.

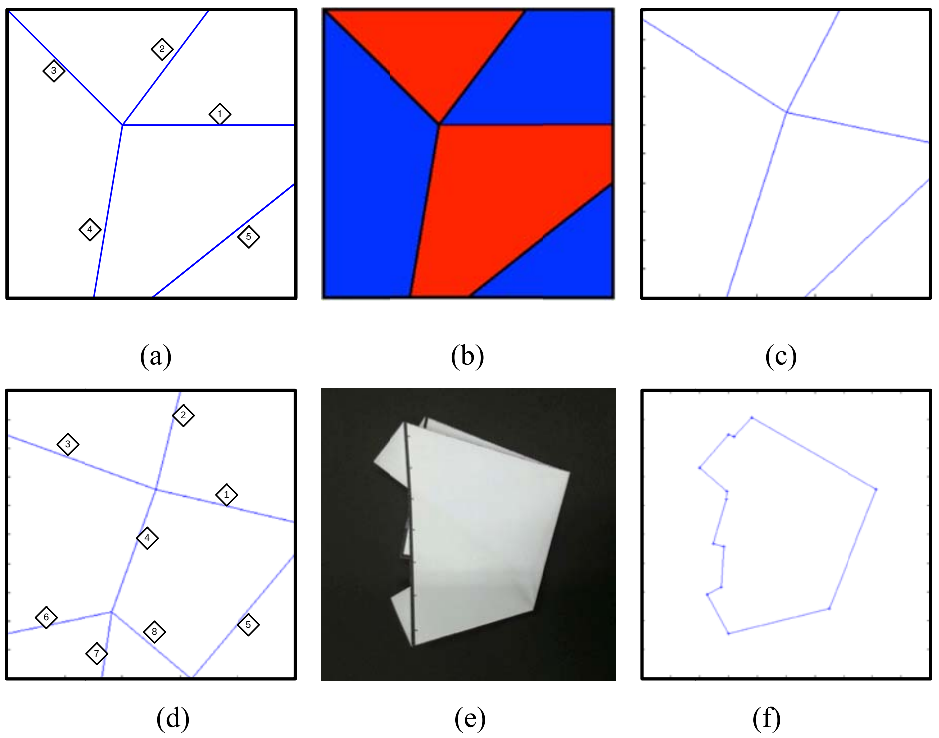


Figure 36 Initial guess pattern used for the first experiment group. (a) The initial guess has five creases; (b) Fill the faces of initial guess with blue and red; (c) A replica of the initial guess that will be used by CEEFOOD to initiate the evolution; (d) The design #1 with the crease numbers from 1 to 8; (e) Manual folding model of the design; (f) The flat-folded profile of the design #1 according to the folding of (e)

Based on the initial guess, CEEFOOD lead the family to evolve into a final design of Figure 36(d), whose flat-folded state profile shape is shown in Figure 36(f). Figure 36(e) is the manually folded model of Figure 36(d). Figure 37 illustrates the step-by-step procedure of folding the crease pattern Figure 36(d) into the flat-folded state in Figure 36(e). Figure 36(f) is the outline of the shape in Figure 36(e). Compared to the initial guess, the design has three more creases that split from the number 4 crease. And one of the branches (the number 8 crease) will be connected with the number 5 crease, which was originally disconnected with others in the initial guess. The three new creases partition the two largest origami faces into four divisions, so that the flat-folded state profile will become closer to 5/18. In all, this design has an overall fitness value of 0.0014, with a flat-folded profile area of 0.2803 and a potential energy evaluation of 0.0011. These evaluations have been improved based on the initial guess Figure 36(c) through CEEFOOD.

Compared to the first group, the second group doesn't apply any initial guesses, just as in the solution generation by the GA with "ice-cracking" presented in the last section. Here, the GA initializes the evolution by using a randomly generated first generation. Therefore, the evolved designs will also be arbitrary. Two of the designs derived are listed in Figure 38.

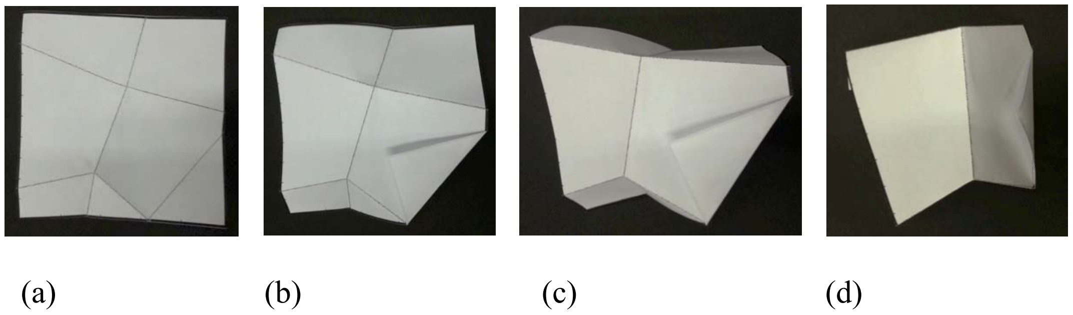


Figure 37 Folding procedure for deriving the folded shape in Figure 36(e) from its crease pattern Figure 36(d). (a) Print out the crease pattern; (b) Fold the lower right corner inward along crease #5; (c) Creases other than #5 are linked, thus must be folded simultaneously. (d) The creases are further folded, and the right half portion of the origami sheet will be folding underneath the left half portion. When all the creases are folded to $\pm 180°$, the shape will become to the state shown in Figure 36(e).

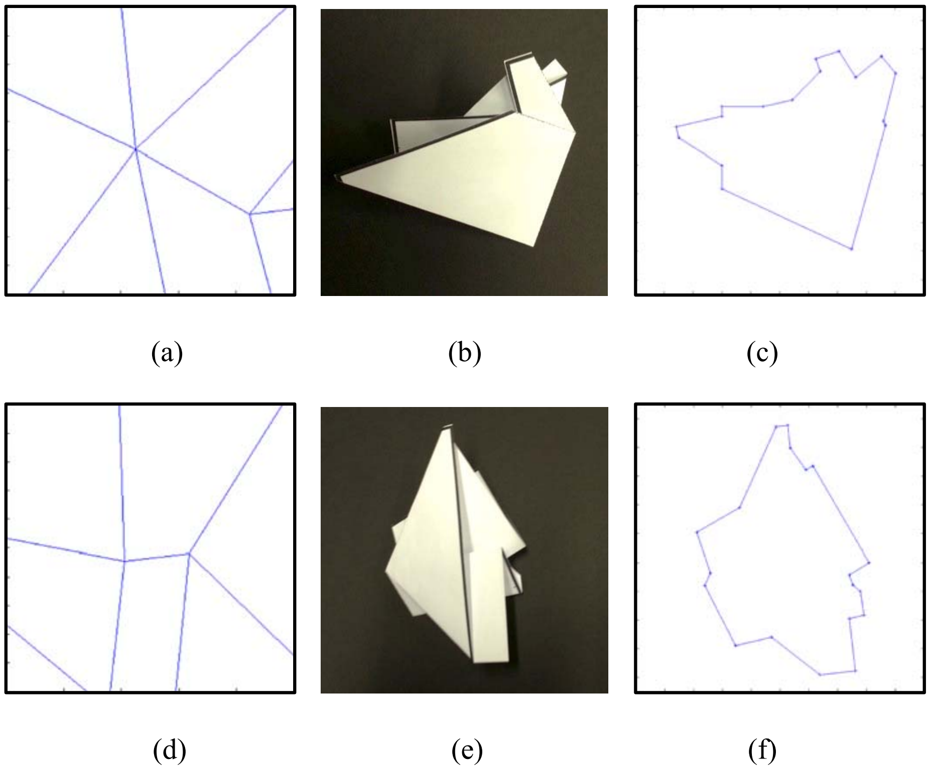


Figure 38 Two crease pattern designs from the second exercise group, which uses randomly generated initial generation of candidate solutions. (a), (b) and (c) present the design #2 and its flat-folded profile; (d), (e) and (f) present the design #3 and its flat-folded profile.

The design Figure 38(a) has 2 interior vertices, 9 creases, and 8 faces. The crease arrangement is made up by 6 longer creases intersecting at one interior vertex, and three shorter creases that branch out from one of the longer creases. Figure 38(b) is a hand-made folded model according to the crease pattern. The flat-folded state profile shape is shown in Figure 38(c), which is also the outline of the shape in Figure 38(b). This design's fitness value is 0.0026, with a flat-folded state profile area of 0.2712 and potential energy evaluation of 0.0010.

The design Figure 38(d) has two connected interior vertices as well, and each of the vertices has four creases. One more crease cuts the largest lower-left face. Figure 38(e) is a hand-made folded model according to the crease pattern. The flat-folded state profile shape is shown in Figure 38(f), which is also the outline of the shape in Figure 38(e). This design has a fitness value of 0.0073, with a flat-folded state profile area of 0.2962 and potential energy evaluation of 0.0035.

Discussion

The fitness evaluations of the three designs are listed in the following Table 3. The table shows the flat-folded state profile area (FFA), the potential energy evaluation (PE), and the overall fitness value (OFV) of each design. Smaller PE and OFV values and the FFA values closer to 5/18 (0.2778) characterize a more fit design with respect to the requirements defined by the problem statements in Table 3. Among the designs, the design #1 has the best FFA and OFV, as well as the second best PE. Aside from the design #1, the design #2 has the smallest PE, and the second best FFA and OFV.

Table 3 The fitness evaluation of the listed designs by the GA with PMR

Design #	**FFA**	**PE**	**OFV**
1 (Figure 36(d))	**0.2803**	0.0011	**0.0014**
2 (Figure 38(a))	0.2712	**0.0010**	0.0026
3 (Figure 38(d))	0.2962	0.0035	0.0073

The overall fitness values of the best candidate solutions (topmost elites) through generations for three designs are plotted in Figure 39. We have used the same CEEFOOD parameter settings (the mutation rate, the crossover approach used, and the proportion of candidate solutions that involve crossover). According to Figure 39, the rates, in which the overall fitness value of the topmost elite decreases along the timeline through evolutionary generations, are different across the three runs. But it is true for the three cases that if the evolution starts with a first generation of lower overall fitness values, it will be more likely that the CEEFOOD evolution will converge faster.

Comparison of the designs reveals the features of CEEFOOD method, which might include both the advantages and shortcomings.

The origami design problem has been proved NP-complete, thus it is impossible to testify the existence of an ideal optimum, or to confirm how many optima exist in the constrained search space. So, in this research, the CEEFOOD won't be designed to derive "the optimum". It accelerates the solution search for the origami design problem that synthesizes the finished shape design. With an initial guess, CEEFOOD generally tends to search for optima close to that initial guess; while without initial guesses, CEEFOOD will finish obtaining arbitrary optima.

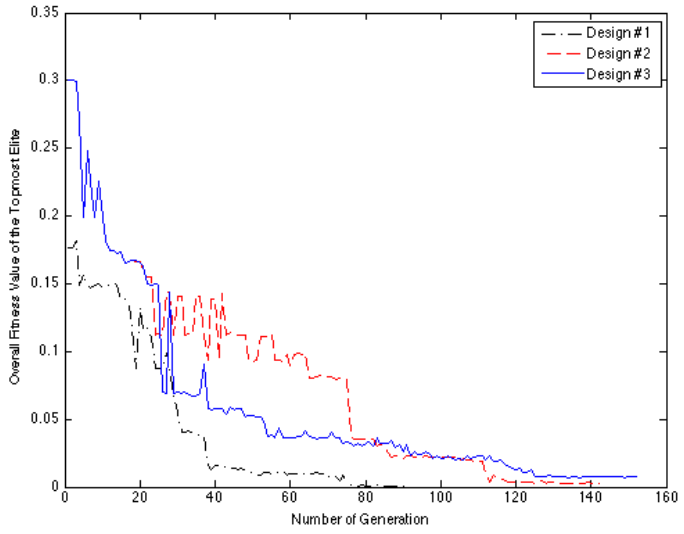


Figure 39 The overall fitness value of the topmost elite through generations

For CEEFOOD, repeated runs may be used to give a more suitable result. As the problem given in this study, the CEEFOOD has derived several designs that can all satisfy the same design requirements. And according to Table 3, design #1 outperforms the others in overall fitness value. If we only trust in raw numbers, design #1 is no doubt the best choice. But it is not always true for the designers to select design #1 in all situations. For instance, if the designer has the preference of a flat-folded state profile shape approximating a triangle, design #2 is better. Nevertheless, we can hold one certain conclusion that, in most cases, the designers have to repeat the CEEFOOD using different initial guesses so as to obtain a pool of candidate designs to choose his most favorable design from.

In many cases, the origami designer doesn't have the complete understanding of the entire search space of a design problem, thus at the beginning he cannot always formulate a "perfect" objective (fitness) function so as to direct the CEEFOOD to derive "the solution". But he can make several runs of CEEFOOD to generate large varieties of solutions, even through his requirements are not ultimately defined. Then through the analysis of the derived designs so far, the designer is able to gradually refine the objective function or form new requirements. An example is as mentioned in the last paragraph. After comparing the designs #1~#3, the designer found the triangular folded state profile advantageous for his applications. In this circumstance, he can add to the objective function (4.1) a new component, which provides an estimation of how similar the folded state profile is with a quadrilateral triangle. Then build a new initial generation including the design #2, and let the CEEFOOD search for finer designs with the newly defined objective function.

Another one of the most effective ways that enables the CEEFOOD to directly regulate the trend of evolution – other than to modify the objective (fitness) function - is to import initial guesses. The analysis Figure 39 shows that a better starting point that initialized the evolution will shorten the number of generations before the CEEFOOD converges. And compared to a randomly generated first generation, some pre-defined initial guesses can be controlled by the users and are more likely to have better fitness values. One more thing to mention here is that the import of initial guesses doesn't have to happen in the first generation of CEEFOOD evolution. Good designs can be introduced into the family at any generation to replace some existing individuals.

Summary

According to the design results and discussions provided above, the GA with “ice-cracking” and the CEEFOOD have both shown their ability to solve the open origami design problem. But they have two essential differences other than the different geometric representations applied.

One difference is in their efficiency of interpreting the genetic representation of each candidate solution. The mapping from the genetic representation to the physical representation of an “ice-cracking” solution cost time proportional to the number of the vertices and creases. On the other hand, the mapping from the genetic representation to the physical representation of a PMR solution requires the density-based clustering and progressive crease update during the clustering and finalization stages of the CRA. Therefore, the time cost will be at least proportional to the product of the number of cells in the PMR and the number of updates on creases. Therefore the CEEFOOD will cost much more time than the GA with “ice-cracking”. If the design problems don’t require flat-foldability in final designs, the time cost of the GA with “ice-cracking” won’t change much, but the time cost of the CEEFOOD will decrease greatly, because the most time-consuming finalization stage of the CRA is omitted for crease patterns without foldability requirements.

Another difference is about their specialization on different classes of design requirements. As the “ice-cracking” representation seamlessly includes the same basic foldability features that can be mathematically defined, the GA using “ice-cracking” is better suited for origami design problems with foldability requirements. The PMR is compatible with structure design tools, thus the CEEFOOD can also include the design of material, the determination of 3D dimensions, and the inspection of structural strength; on the contrary, the GA with “ice-cracking” is more competent as a computational geometric design method.

CHAPTER VI

EXTENSIVE APPLICATION ON ORIGAMI LIQUID CONTAINER DESIGN

In this chapter, the GA adapted for origami design will be applied for a problem of deriving an origami water container. With all the considerations that have been discussed in previous chapters, the water container design problem also requires the design of the crease folding angles and the exclusion of any 3D face intersections, though the upper level of the GA method doesn’t change much. This demonstrative problem is presented to support the broad applicability of the GA on origami design.

Problem Statement

The design problem asks for a folding of a non-leaking water container. The description is as below.

Objective: Given a 1-by-1 square origami sheet, design the crease pattern and the folding angles of the creases to get a water container with a volume of 0.02.

Conditions / constraints:

1. The origami folding is rigid. Faces should not be bended, crumpled, cut, or torn during and after the folding procedure.

2. The origami sheet is very thin. The thickness of the material doesn’t need to be considered during the calculation of container volume.

3. When calculating the volume, one must place the folded origami container on a flat plane. And with locked crease angles, the folded shape must be able to stand. This requires that the container cannot be a cone or an upside-down pyramid.

4. In addition to the requirement on the volume, a lower degree-of-freedom (DOF) and shape complexity are also preferred. The DOF here means the DOF of the origami structure as a multi-sheet mechanism. The shape complexity measures the number of creases and faces. More creases or faces indicate higher shape complexity.

Therefore, as the volume is the only primary factor that affects the quality of a design, the objective function – which is also the fitness function - is simply the difference between the actual volume and the desired volume. And the GA will be applied to minimize this objective function.

$$Fitness = |volume - 0.02| \tag{32}$$

Constraint #4 also requires the minimization of the DOF and shape complexity. But for this problem, we temporarily put the two factors aside from the objective function. To incorporate them into the GA solution searching and evaluating, we put penalty only on the designs, which have more than 3 DOFs or 25 creases.

Modified "Ice-cracking" for 3D Folding

The introduction of the "ice-cracking" in Chapter III has fully defined how the representation directs the development of creases in an origami crease pattern. But for this application, the GA has to synthesize the folding angle design with the original crease pattern design. Therefore, it's required that the "ice-cracking" representation can also include the information of the candidate folding angle design as well.

In fact, if an origami structure has a known n_{DOF}, the number of the dihedral folding angles that we can freely decide is equal to the n_{DOF}. Defining less or more than n_{DOF} folding angles will result in the mechanism to be under-constrained or over-constrained. And the n_{DOF} folding angles cannot be assigned to any arbitrary n_{DOF} creases. For an interior vertex with n_{NOC} creases linked, the number of defined creases should not exceed $n_{\mathrm{NOC}} - 3$. In this research, we call the set of n_{NOC} creases, whose folding angles can be freely determined and can also work together to determine the folding angles of all the other creases, the set of *actuator creases*.

The "ice-cracking" representation is advantageous in making the selection of actuator creases very easy. We only need to make sure that after each "ice-cracking" step, the resultant crease pattern has just enough actuator creases, so that the folding angles of all the existing creases are defined. In the initialization step, an interior vertex with n_{NOC} creases is created. There are $n_{\mathrm{NOC}} - 3$ actuator creases that can be arbitrarily chosen among the n_{NOC} new dummy creases. In a forking step, an interior vertex with n_{NOC} creases is located on an existing dummy crease. Since the dummy crease can either be an actuator crease, or be passively actuated, it can be treated as a local actuator crease for the new vertex. So the other $n_{\mathrm{NOC}} - 4$ actuator creases can be arbitrarily chosen among the $n_{\mathrm{NOC}} - 1$ new dummy creases. In a resolution step, an interior vertex with n_{NOC} creases is located on the intersection of m_{inter} existing dummy creases. Since the m_{inter} dummy creases can either be actuator creases, or be passively actuated, they can be treated as local actuator creases for the new vertex. So the other $n_{\mathrm{NOC}} - 3 - m_{inter}$ actuator creases can be arbitrarily chosen among the $n_{\mathrm{NOC}} - m_{inter}$ new dummy creases. Otherwise, the mechanism will be over-constrained. For this research, without causing confusion, we generally select the first few dummy creases that are created in each step as the new actuator creases.

To include the candidate folding angles of the actuator creases into the "ice-cracking" representation, we need to append additional bits of the code for each step. For instance, the initialization step for the flat-foldable vertex #1 of pinwheel pattern in Figure 13(a) used to apply the argument set {x_1=0.25, y_1=0.25, n_{NOC}=4, θ=0, κ_1=255, κ_2=255, κ_3=255, κ_4=85}, and the corresponding genetic representation: '00 01110000 01110000 00 00000000 10101010 10101010 10101010 01100110'. Now since we need to define the angles of $\frac{n_{\mathrm{NOC}}}{2}$ actuator creases, we append new arguments φ_i $(i = 1, \ldots, n_{\mathrm{NOC}} - 3)$. Because the vertex #1 needs only one actuator crease, the new argument set for this step becomes {x_1=0.25, y_1=0.25, n_{NOC}=4, θ=0, κ_1=255, κ_2=255, κ_3=255, κ_4=85, φ_1=$\pi/12$} and the equivalent genetic representation is '00 01110000 01110000 00 00000000 10101010 10101010 10101010 01100110 01011010'. If more than one new actuator creases are needed in one step, there will be more φ_i's appended to the argument set, and thus more bits added to the genetic representation.

Rigid-foldability and the Dihedral Folding Angles

Rigid-foldability is the key feature of rigid origami. It requires all flexure in the shape to take place only along the creases (Watanabe and Kawaguchi 2006). Comparing to flat-foldability, rigid-foldability has one criterion in common in that every face must keep flat at the unfolded and folded states. But while flat-foldability of an origami shape is only concerned when all the creases are folded to $\pm\pi$, the rigid-foldability deals with the entire procedure of a shape being folded from the initial state (unnecessarily the fully-spread state) to the folded state with desired crease folding angles. Therefore, unlike the flat-foldability, rigid-foldability won't need the folding angles to reach $\pm\pi$ in all the cases, but it requires the existence of a continuous route from the initial state to the folded state, where at any arbitrary timepoint the faces must always keep flat. The rigid-foldability has been studied by many predecessors, including D. A. Huffman (Huffman 1976), T. Hull (Hull 2006), N. Watanabe (Watanabe and Kawaguchi 2006), and T. Tachi (Tachi 2006, Tachi 2010). Tachi has given a thorough solution for checking the rigid-foldability, as well as an incremental approach of calculating the crease angles in his research on a rigid origami simulator (Tachi 2006).

The computational complexity of numerically testifying the rigid-foldability thus has to go through the following recursive stages:

Algorithm: Inspect the rigid-foldability of a crease pattern with pre-defined folding angles

```
1. For each possible folding sequence:
    1.1. Partition the folding procedure into motion frames separated by time
         intervals
    1.2. For each motion frame:
        1.2.1.    Simulate the semifolded shape
```

```
1.2.2.      For each pair of non-linked faces:
    1.2.2.1.      For each crease in one of the two faces:
        1.2.2.1.1.  Check whether the crease intersects with the other face
        1.2.2.1.2.  If there exist intersections, the shape is not rigid-foldable
                    with the current folding sequence. Then go to stage 1.
```

The computational complexity of the prior algorithm depends on the number of possible folding sequence (n_{FS}), the number of motion frames (n_{MF}), the number of faces (n), and the average number of creases bounding each face (m_F). And the estimation of the complexity would be $O(n_{FS} \cdot n_{MF} \cdot n(n - n_F) \cdot m_F)$, where n_F is the average number of linked faces for each face.

The number of possible folding sequences can be large. We define fully folding a crease or a set of creases as folding the crease from flat to their final angles, and partially folding a crease as folding the creases from flat to angles larger than 0 but smaller than the final angles. For an origami shape with n_{DOF} degrees of freedom, the shape has (n_{DOF}!) possible folding sequence, if the creases are fully folded consecutively. Therefore, given the crease pattern Figure 40(a) with 7 parallel creases and its desired folding angles (not specified here), we can derive a folded shape of Figure 40(b). Suppose that the folding sequences can only be formed of stages of fully folding the crease sequentially. So the folding sequence of the 7 simple folds must follow a rule that the creases marked by larger numbers can only fold after all the creases with smaller numbers are folded. Therefore, the shape defined in Figure 40 has $7! = 5040$ possible folding sequences, among which there are $6 \times 4! = 144$ legal ones. To prove rigid-foldability, one needs to try the 5040 possible folding sequences one by one until he arrives at a legal one. In addition, if we allow a folding sequence to include partially folded creases in intermediate stages, there will be infinitely many folding sequences that need to be simulated and tested so as to prove the rigid-foldability of the shape.

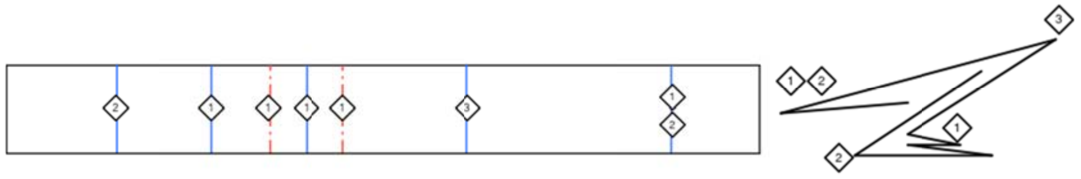


Figure 40 The rigid-folding of a long strip with 7 parallel creases. (a) The crease pattern; (b) The side view of the folded shape.

In this research, we aren't able to afford an attempt at every folding sequence for arbitrary designs. So as a compromise, in this application, we directly inspect if the shape can be folded according to the folding angles, and use the existence of face intersections to tell if the design is not rigid-foldable.

The starting step of inspecting rigid-foldability of a design is to calculate the folding angles at the folded state. In each design generated by GA, only n_{DOF} of the creases have been assigned folding angles. So we need to calculate the folding angles of all the other creases using an equation given in (Tachi 2006):

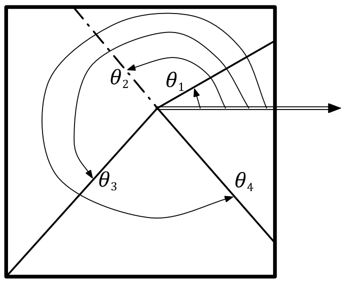


Figure 41 A simple single vertex crease pattern showing the definition of angles. The double-line arrow is the vector of the baseline, which in this case overlaps with the Cartesian x-axis

$$For\ each\ interior\ vertex: \prod_{i=1}^{n_{\mathrm{NOC}}} \chi_i = I \tag{33}$$

$$\chi_i = \begin{bmatrix} cos^2\theta_i + cos\rho_i \cdot sin^2\theta_i & (1 - cos\rho_i)cos\theta_i sin\theta_i & sin\rho_i sin\theta_i \\ (1 - cos\rho_i)cos\theta_i sin\theta_i & sin^2\theta_i + cos\rho_i \cdot cos^2\theta_i & -sin\rho_i cos\theta_i \\ -sin\rho_i sin\theta_i & sin\rho_i cos\theta_i & cos\rho_i \end{bmatrix}$$

where χ_i is the transformation matrix with respect to the i-th crease of the vertex, n_{NOC} is total number of creases of the vertex, θ_i defines the orientation of the i-th crease with respect to the vertex (as shown in Figure 41), and ρ_i is the folding angles of the i-th crease. In the equation, all of the θs and some of the ρs are known. And we use numerical methods to compute the values of all the unknown ρs in the non-linear equation. In order not to cause any confusion between the definitions of folding angle ρ_i and the dihedral angle φ_i of the two faces sharing the i-th crease, we define

$$\rho_i = \begin{cases} \pi - \varphi_i\ ,\ if\ the\ i - th\ creases\ is\ a\ valley \\ \varphi_i - \pi\ ,\ if\ the\ i - th\ creases\ is\ a\ mountain \end{cases}. \tag{34}$$

We will solve the equation for every interior vertex until the folding angles of all the creases are derived. Then we can confirm the final orientation of the faces and then inspect if there are intersections among them. If there are intersections, the shape is not rigid-foldable, and its fitness value is set to 1, which indicates an illegal candidate solution.

Without the incremental calculation of the crease folding angles from the initial state to the folded state, we have saved lots of computational efforts, but we have also left the rigid-foldability partially proved. This results that even if there is no face intersection found through the process described above, the shape is still possibly not rigid-foldable. If the total DOF of the design is 1, we can simulate the whole deployment procedure from the folded state to the initial state, or simulate the whole folding procedure from the initial state to the folded state. But as the number of DOF increases, the computational complexity rockets exponentially. Even if we only simulate for the candidate designs without face intersection at the folded state, it's still not a practical solution.

Nevertheless, since this is a demonstrative problem, we temporarily avoid the proof of the theoretical rigid-foldability, and leave it to future discussion and research. As we don't want to restrict the DOF of the designs, we decrease the search space by setting an upper limit of 30° for the folding angle value of each *actuator crease*. The actuator creases are the set of creases, whose folding angles can determine the folding angles of all the other creases. The number of actuator creases are equal to n_{DOF}. With small final folding angles, the shapes are less likely to have collision among faces during the folding procedure, if we can prove a priori that the shape has no face intersection at the folded state. A "small" folding angle is only quantifiable for an explicitly defined shape. In general, it means the corresponding angle won't be folded too sharp to cause any faces to get very close to each other either at the final folded state or during the folding procedure. Such simplification on rigid-foldability inspection doesn't really guarantee every design that the GA generates to be rigid-foldable, but it considerably lowers the probability of deriving non-rigid-foldable shapes.

Having the folded shape, the computational complexity of checking the intersections between faces is $O(n^2)$. The algorithm examines each pair of faces that are not directly linked by a crease. If any crease of one face intersects with the other face, then the two faces also intersect with each other.

Algorithms for Calculating the Volume

Other than the encoding of actuator crease folding angles and the inspection of rigid-foldability, the third new function the GA needs to include is the calculation of the volume of each candidate solution it generates.

At the first step, the volume calculation algorithm should have to confirm the existence of a flat bottom that was requested in the problem constraint #3. In theory, we can always find a way for the shape to stand, as long as there are three vertices on the convex hull of the folded shape (no matter whether any of the three vertices are directly linked by creases or not), and the three vertices form a 3D triangle that have no intersection with other faces. But including such situations would make the problem complicated. Therefore, for my research so far, I only consider two types of container bottoms. One type uses a face as the container's bottom as Figure 42(a); the other applies two linked creases that form a V-shape bottom support to let the whole shape stand on it as Figure 42(b). As shown in the two figures, the yellow origami container has a flat bottom, while the green one with the V-shape also has a V-groove as its internal space.

Now, let's think about which portions of the folded shape contribute to the container's volume. Here, suppose that we can inject water to every location inside the folded container. We define the amount of water held in the container - if we use the above way to inject water over an infinitely long duration – as the volume of the container.

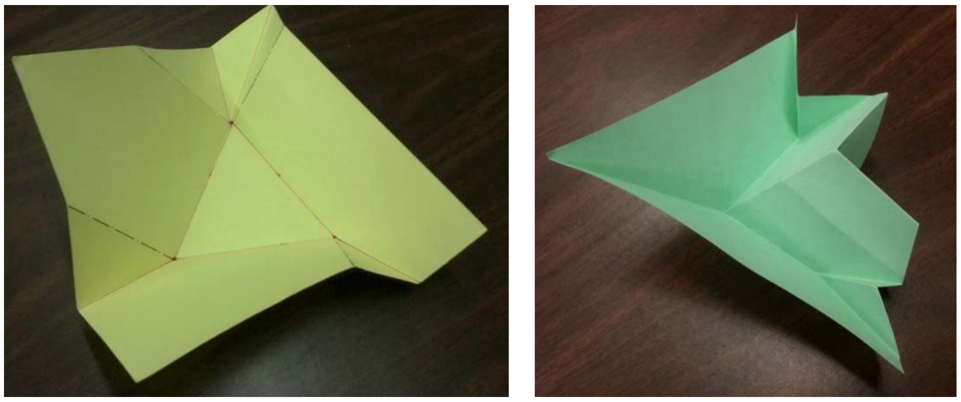

(a) (b)

Figure 42 Two hand-make origami shapes that can be used as water containers. (a) A shape uses the triangular face in the middle as it bottom. (b) A shape uses its two consecutive creases as its bottom.

The most widely used algorithm for calculating the volume of a polyhedron is to partition the shape into tetrahedrons or pyramids. The sum of the volumes of the tetrahedrons or pyramids will be the volume of the whole shape. But in this research, it is not easy to confirm the actual shape of the water held within. Therefore, we turn to the algorithm of calculating the volume as described below. It applies a numerical approach to derive an approximation of the volume value we want.

Algorithm: calculate the volume of a folded container

```
1   Derive the plane $P_0$ where the bottom is in. Make sure that the normal of $P_0$ is
    upward.
2   If the bottom is a face, then $A_0$ is the area of the face; else, $A_0 = 0$.
3   Create and initialize $V = 0$
4   Define the precision of volume calculation as $V_p$
5   While $P_i$ has intersection(s) with the shape
    5.1   If $A_{i-1} \neq 0$, $h_{i-1} = V_p / A_{i-1}$; else, $h_{i-1} = V_p / \max(A_0, A_1, \dots, A_{i-1})$
    5.2   $P_i$ is a parallel displacement of $P_{i-1}$ with distance $h_{i-1}$
    5.3   Look for the intersection segment $e_j^i$ of $P_i$ with all the faces
    5.4   Look for closed loops formed by $e_j^i$
    5.5   Create and initialize $A_i = 0$
    5.6   If there are closed loops
          5.6.1 For each closed loop, $A_i = A_i$ + area of the polygon defined by the
                closed loop
    5.7   $V = V + (A_i + \sqrt{A_i \cdot A_{i-1}} + A_{i-1}) \times h_{i-1}/3$
    5.8   The resultant V is an approximation of the container's volume.
```

In the algorithms above, the shape of the folded container is partitioned into slices. If a slice is a complete ring or if it has several rings, the volumes of the rings will contribute to the total volume. The computational complexity of this algorithm depends on the nominal volume of the shape. Larger volumes result in more slices. For the i-th slice with n_e^i intersection segments between the section plane and the origami faces, the computational complexity of deriving its volume is $O(n_e^i) + O(n_e^i log n_e^i)$, which is $O(n_e^i log n_e^i)$. The linear component $O(n_e^i)$ of the complexity is for the derivation of closed loops (the formation of section polygons) from n_e^i segments. The other component $O(n_e^i log n_e^i)$ is for the triangulation of the section polygon(s) and the calculation of the area of each triangular.

Design Results and Discussion

After confirming the way for the genetic representation of candidate solutions, the algorithms to calculate the volume of a rigid-foldable shape, as well as the fitness function, the GA could be applied directly on the problem.

At first, we give a fundamental analysis of this problem. We want to know whether the target volume is achievable by a folding of the 1-by-1 origami sheet. Using intuitive guess, we can manually derive a crease pattern as the one shown in Figure 43(a). In this manual design, the creases are arranged symmetrically. The dimensions are given in Figure 43(a). When folded, the container develops into an "ash-tray" build, which has a square bottom of an a-by-a, and a height of $(1-a)/2$. The volume of the container is thus $a^2(1-a)/2$, where $0 < a < 1$. For this design, the maximum volume $2/27$ =

0.0741 can be obtained when $a = 2/3$. Therefore, the target container volume falls within (0, 0.0741], thus is attainable by a folding of this 1-by-1 origami sheet. Then if the target volume is 0.02, the length of the bottom edge can be $a = 0.2276\ or\ 0.9563$.

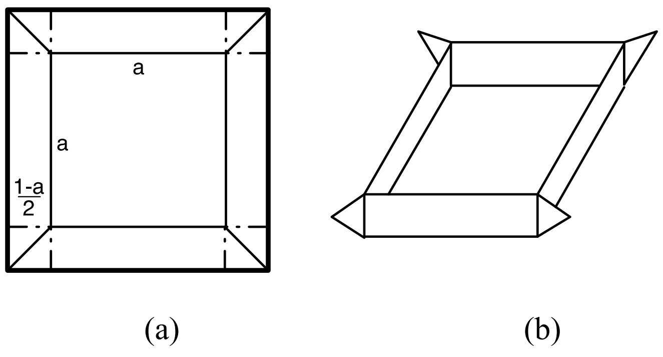


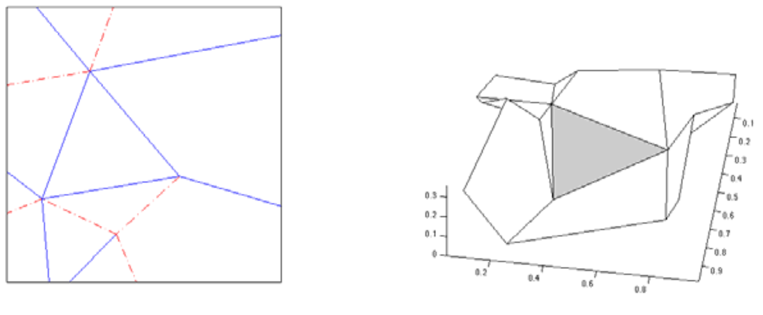
Figure 43 An intuitive manual design of an "ash-tray" style water container. (a) The crease pattern design with an a-by-a square in the middle. (b) The folded shape.

However, the above design is a trivial one. This solution only satisfies one type of container bottom. If the problem prefers the container having no surface contact with the plane where it stands, the trivial solution would not be qualified.

Now in order to have a generic method that works for the type of design problem with more varieties of requirements, we choose the same metaheuristic – the genetic algorithm, which has been testified for flat-foldable origami design problems. The way of incorporating the folding angles into the "ice-cracking" representation has been developed in a prior section. In the next section, I will present some of the designs generated by the GA, and make some comments on them.

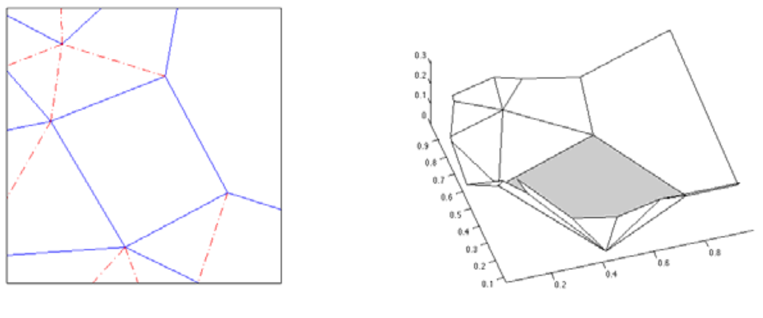
Computational Designs by GA

In this section, the GA with "ice-cracking" is applied to generate several diverse designs. These designs all reach a volume of 0.02 with errors of lower than 0.5%. The crease patterns and the folded shape of 6 designs are listed in Figures 44-49.



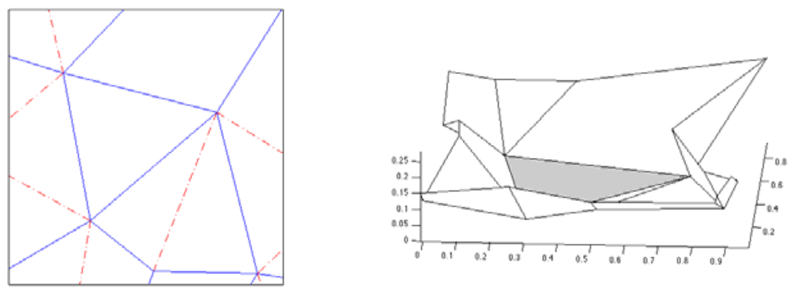
(a) (b)

Figure 44 Design #1 for an origami water container with a volume of 0.02. This design has 12 faces, 15 creases, and 14 vertices. (a) The crease pattern; (b) The folded shape (the grey face is used as the bottom).



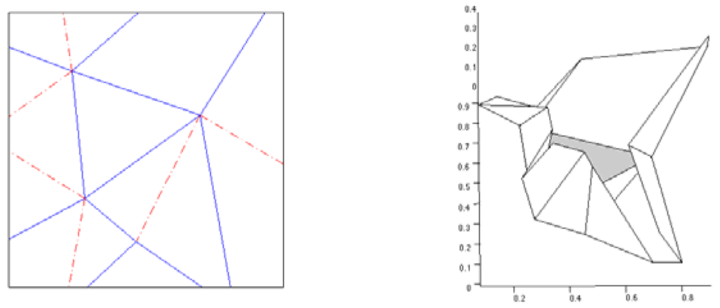
(a) (b)

Figure 45 Design #2 for an origami water container with a volume of 0.02. This design has 17 faces, 22 creases, and 20 vertices. (a) The crease pattern; (b) The folded shape (the grey face is used as the bottom).



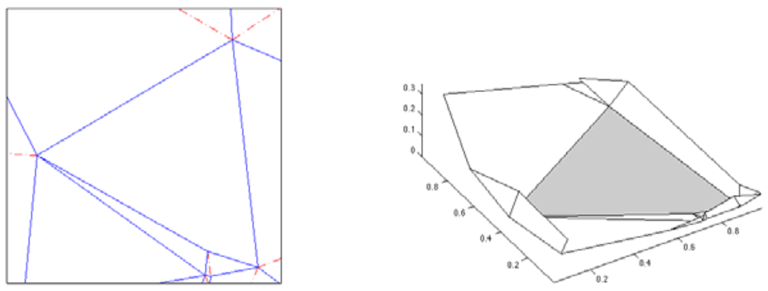
(a) (b)

Figure 46 Design #3 for an origami water container with a volume of 0.02. This design has 17 faces, 21 creases, and 19 vertices. (a) The crease pattern; (b) The folded shape (the grey face is used as the bottom).



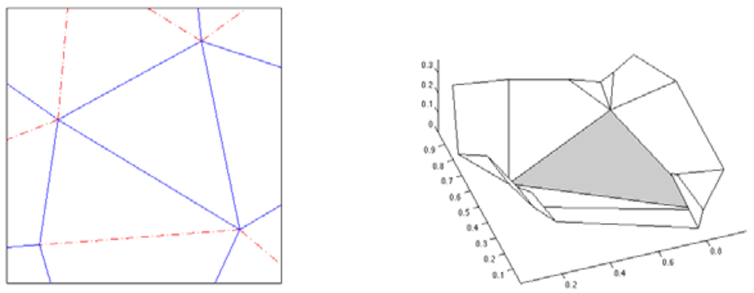
(a) (b)

Figure 47 Design #4 for an origami water container with a volume of 0.02. This design has 14 faces, 17 creases, and 16 vertices. (a) The crease pattern; (b) The folded shape (the grey face is used as the bottom).



(a) (b)

Figure 48 Design #5 for an origami water container with a volume of 0.02. This design has 18 faces, 23 creases, and 20 vertices. (a) The crease pattern; (b) The folded shape (the grey face is used as the bottom).



(a) (b)

Figure 49 Design #6 for an origami water container with a volume of 0.02. This design has 17 faces, 21 creases, and 19 vertices. (a) The crease pattern; (b) The folded shape (the grey face is used as the bottom).

In the above folded shapes of the designs, the faces colored in light grey are the bottom faces of the corresponding origami water containers. The numbers of faces vary from 12 to 18. The numbers of creases vary from 15 to 23, which don't exceed the upper bound of 25 creases set in prior section. The numbers of vertices, including interior and border vertices, vary from 14 to 20.

Discussion

Without a specific engineering application of the origami water container, we don't have the determined criteria to judge which of the above 6 designs and the trivial "ash-tray" design in Figure 43 is the best. But in topological formation, design #1 in Figure 44 has the least number of creases, thus has the least shape complexity. And the design #4 in Figure 47 is also a nice one, since it has a pot-styled folded shape, while others are more shallow and saucer-styled. And in the design #4, the three faces around the lower left corner of the crease pattern (Figure 47(a)) forms a narrowing spout to make pouring out the water from the container more easy.

From the computational designs, we can find another common feature. They all feature small crease folding angles. Let's use a numerical index to evaluate the folding angles. Define a concept named ***folding effort*** U, so that

$$U = \sum_{i=1}^{m} kL_i|\rho_i| \tag{35}$$

where m is the number of creases, k is a scalar coefficient, L_i is the length of i-th crease, and ρ_i is the folding angle of the i-th crease. The folding effort is a linear approximation of total material strain generated by folding along the creases. Of course, the real internal strain is not simply linear to the folding angle, and it depends on at least the material, the thickness of the origami sheet, and the folding radius. But as this research mainly focuses on the origami design metaheuristic, we leave the estimation of the actual strain to future work.

Using the equation with $\mathrm{k} = 1$, the folding efforts of the 7 obtained designs are shown in Table 4. According to Table 4, designs #1 and #6 have the least folding efforts, while the “ash-tray” has the greatest folding effort. For achieving the same volume, different designs have large differences in their folding efforts. We can also make a reasonable prediction that if using the actual structural strain to replace the folding effort, designs will still have highly diverse values of strain generated through bending of the material.

Table 4 Folding efforts of the origami water container designs

Design	Folding Effort
“Ash-tray” Figure 43	12.1500 ($a = 0.2276$) 12.5428 ($a = 0.9563$)
#1 Figure 44	4.9213
#2 Figure 45	5.9785
#3 Figure 46	6.1743
#4 Figure 47	8.5889
#5 Figure 48	5.3367
#6 Figure 49	4.8035

The folding effort can be simply calculated based on the geometry. But the structural strain may require finite element analysis tools that generate more cost on computing. However, the FEA is realizable computationally, thus it makes the structural strain by folding a compatible performance index of the genetic algorithm. This specific extension remains future work.

Aside from comparing the designs on their performance, it is also necessary to inspect the progress of how they are developed by the GA. I have made a plot below to display the topmost designs that are optimized through generations in the families of the 6 designs.

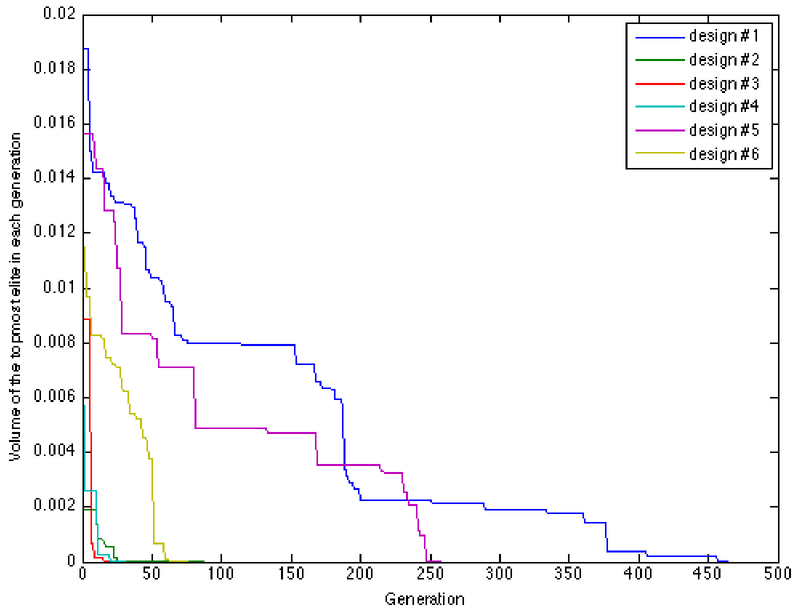


Figure 50 The volume of the topmost elite across generation of the 6 designs produced by the GA.

The 6 different runs of the GA apply the same set of parameters for the GA evolutionary operators: the size of the family is constantly kept at 30 for each generation; the length of the genetic representation is 700 bits; 18 new candidate solutions will be generated in each generation by 4-point crossover; each bit of the genetic representation of the candidate solutions has a probability of 6% to get flipped; 6 new random candidate solutions will be added to the pool; the captive breeding's external crowd has a size of 20; 3 of the best captured elites solutions will be brought back into the current generation; the atavism will look back 20 generations for 1 ancestor elite.

However, with the same setting of the evolutionary operators described as above, several runs of the GA converges to very different solutions through different numbers of evolutionary generations. Among the 6 designs, design #1 takes more than 450 generations; design #5 takes more than 250 generations; design #6 takes more than 50 generations; but the designs #2, #3, and #4 take only about 25 generations.

One reason that the time cost of the convergence of the GA is firstly extended due to the highly nonlinear solution space of the origami problem. The actual n-to-1 genotype-phenotype mapping defined by the "ice-cracking" also expands and deforms the solution space into a more complex design space. In addition, the randomly generated first generation of candidate solutions that initializes the GA may be located far from an optimum in search space, thus making the evolutionary optimization procedure longer.

Nevertheless, a common feature revealed by the generation of the 6 designs is that a better first generation will generally result in faster convergence. The families of the designs #2, #3, and #4 converge to their corresponding final designs faster, because the topmost elites in their first generations are more optimal than those from the families of the designs #1 and #5. The same conclusion can also be found in chapter 5, when I compared the flat-foldable origami designs derived by the GA with PMR.

Summary

This chapter has presented an extensive application of the GA with “ice-cracking” on origami design problems that ask for geometric and functional requirements at the 3D folded state of the origami.

As the application synthesizes the folding angles of the creases, the “ice-cracking” representation needs to include the declaration of the folding angles of the actuator creases. Moreover, this chapter also makes some discussion about the rigid-foldability and the calculation of the volume of a folded origami container.

Since an arbitrary origami design has unpredictable degrees-of-freedom and infinitely many possible folding sequences to reach the desired folded state, the inspection of rigid-foldability has to be simplified. My solution is to limit the maximum folding angles and just confirm the legality of the final folded state. For folding a water container, using smaller folding angles is reasonable, since we don’t anticipate a design, where faces intertwine with each other very closely. Then with smaller folding angles, a legal rigid folding procedure is more likely to exist for a design as long as its final folding state is legal. The legality of a folded state depends on the crease pattern design and the assignment of crease folding angles. For a given crease pattern and crease folding angles, we can simulate the semi-folded state of it. If face intersections exist in the semi-folded state, then the design is not rigid-foldability.

When calculating the volume of the origami container, the classic polyhedron volume calculation algorithm is not applicable, since the shape of the water that is held by an arbitrary origami container is not explicitly defined. In this chapter, I have introduced my algorithm of deriving the volume. The algorithm uses a plane parallel to the bottom to sweep the folded container. The sweep planes section the container shape into slices. If a slice includes rings, which are circumference by fillets of sectioned faces, and the rings have the water-contacting side inside, then the volume of the rings adding their two caps will be counted as a portion of the container's volume.

CHAPTER VII

CONCLUSION AND FUTURE WORK

Contributions to Origami Engineering and Design

This research creates and applies abductive methods for origami design. This dissertation runs through three applications of genetic algorithms on the design of flat-foldable origami structure and rigid-foldable containers. The research has made the following contributions to origami engineering and origami design:

1. This research has created a design method framework for solving the open origami design problems. In this dissertation, the genetic algorithm and its variation – the evolutionary computational embryogeny – are used as the prototype for constructing and adapting the design method framework. The capability of searching in space that is hard to be explicitly and fully modeled makes the GA-based meta-heuristics especially advantageous for the open origami design problems, which has been proven NP-complete in this dissertation. Unlike the traditional origami design problems that look for the paper-made replications of some desired shapes, the open origami design problem won't define the exact desired shape or the specific topological structure that the desired shape will have. The open origami design problems ask for the achievement and/or the optimization of geometric and functional requirements on the crease pattern, the folding procedure, and the folded state of the origami designs. Since open origami problems theoretically

accept any arbitrary designs, the search space should ideally cover any design as well. This fact also causes the open origami problems NP-complete.

2. This research creates a direct geometric representation – "ice-cracking" – to relate the encoding of the genetic representation and the definition of the physical representation of origami designs processed by GA. The genetic algorithm with direct "ice-cracking" representation is also testified on an origami design for folding a water container. The application has proven that the crease folding angle design is also achievable. Thus 3D shape design is available for GA with "ice-cracking".
3. Computational evolutionary embryogeny (CEE) is a variation of GA that applies encoded generative rule sets as the genotype of an origami design. This research adapts CEE for open origami design problem by defining an indirect geometric representation – pixelated multicellular representation (PMR) – for origami designs. PMR is defined as a third equivalent representation to the genetic code (genetic representation) and the crease pattern (physical representation) of an origami design.
4. The crease restoration algorithm (CRA) is created to extract the origami crease pattern from a pixelated image. The CRA is applied to relate the PMR and its crease pattern.
5. As a part of fitness evaluation, new algorithms that are compatible with GA are developed to solve the MV-assignment, global flat-foldability, and rigid-foldability problems for a given crease pattern.

Conclusions

In this dissertation, I have created a design method framework for the open origami design problems that requests the creation of origami shapes with anticipated geometric and functional requirements. The type of problem is described as “open”, because the problem statements don’t strictly define the final folded shape of an origami. Instead, these open problems have design objectives and constraints on the geometric and functional performance, such as the flat-folded state shape area, the location of the center of mass, or the volume. Without a known shape or at least a deterministic topological formation of the desired origami designs, the methods have to search over the design space that cover all arbitrary origami shapes. As the design space including any arbitrary crease pattern is large and nonlinear, and the origami design problem has been proven NP-complete, I instead turn to the abductive methods.

In this dissertation, I implemented two specific abductive methods that are the genetic algorithm and its variation – the computational evolutionary embryogeny. The two methods both feature a specifically designed geometric representation for candidate origami designs. There are three key aspects that are needed to clarify prior to the implementation of the genetic algorithm. Except for the fitness function, this research focused on the design of a geometric representation that provides the framework of defining both the genetic representation and the physical representation of an origami candidate design, and on the development of basic and complementary genetic operators to accommodate the properties of the design space.

One geometric representation I designed for the GA is the "ice-cracking". The "ice-cracking" representation is used to generate, encode, and describe origami crease patterns. The "ice-cracking" essentially presents a systematic sequence for ordering all the vertices and creases in a growing manner similar to the natural process of ice-cracks emerging and developing on an icy surface.

The "ice-cracking" is executed through a procedure consisting of an initialization step (locating the first vertex), forking steps (locating a vertex along one existing dummy crease), resolution steps (defining a vertex at the location of the intersection of two existing dummy creases), and a finalization step (converting all the remaining dummy creases to creases). Each step involves three basic operations: creating a new vertex, fixing creases, and placing dummy creases. The static formation of the "ice-cracking" representation enables its steps or operations to be defined by sets of arguments with fixed formats. The sets of arguments could then be converted to Gray code strings, which are used to encode the entire "ice-cracking" for any crease pattern. In this chapter, I have focused on flat-foldable origami, and set up a grammar for encoding "ice-cracking" so that any Gray code string is capable of being translated to the "ice-cracking" that defines a flat-foldable crease pattern.

Another geometric representation is called pixelated multicellular representation (PMR). The PMR defines an origami structure by distributed cells. The cells fundamentally work as conceptual indicators, which directly define the origami by coloring the faces in the same way that LEDs in a LED matrix screen display a picture. Equivalent to an LED, each cell will have three key properties – cell type that shows the color, cell size that indicates the light intensity, and cell position that implies the resolution of cells. Every collection of cells with the same type (color) will define an origami face. The creases are linear classifiers generated to separate faces. I proposed a crease restoration algorithm to extract the equivalent crease pattern from a pixelated multicellular representation. The PMR is advantageous for its availability of expressing any arbitrary simple or non-simple creases. But after discretizing the cell properties for the computational applications, the representativeness of a pixelated multicellular pattern depends on the effective resolution and effective precision, which are the metrics relative to the setting of two of the cell properties – cell position and cell size. A third cell property is the cell color. In this research, the 2-colorability feature of flat-foldable origami has enabled the cell color to be chosen between minimally two colors.

As the two geometric representations combined with the GA, they will form two effective abductive design methods for solving open origami problems.

For GA with "ice-cracking", the genetic algorithm (GA) could consequently take advantage of the feature of "ice-cracking" that any single network flat-foldable crease pattern can be encoded and any arbitrary Gray code can represent one crease pattern. In order to cope with the large and non-linear design space of origami design problems, I adapted the GA through balancing the preservation of the elitism and diversity. On one hand, to protect the elite individuals, I directly lower their probability of getting mutated. Moreover, I also introduced two mechanisms called captive breeding and atavism. The captive breeding mechanism captures and archives good individuals from the evolving generations, and occasionally releases some back to a later generation. The atavism lets the time flash back, so that some of the elite individuals from an earlier generation can be introduced into the current generation. For diversity, on the other hand, I raise the mutation rate for every individual, and apply hybrid fitness ranking, in order to prevent the premature convergence of extreme elitism. The adapted GA with “ice-cracking” has been implemented and testified on an origami design problem that optimized several geometric and functional properties simultaneously.

Compared with the GA with “ice-cracking”, the CEEFOOD method utilizes a hybrid coding method to define the alternative PMR of each candidate solution. The hybrid code is composed of an explicit coding part describing a PMR embryo and an implicit part encrypting a set of generative genetic rules, which are used to guide the embryo to develop into a mature state. The CEEFOOD also combines the attainment of MV-assignments and face overlapping orders in its fitness evaluation stages. During the evolution stage, CEEFOOD cooperates with the elite preservation mechanism and code assimilation to support the progressive adaptations among proposed candidate solutions. The synergy of these mechanisms allows CEEFOOD to employ a high mutation rate without frequently eliminating already emerged good results.

In addition, by introducing initial guesses, the evolutionary trend of CEEFOOD will converge close to a good initial. On the contrary, using a totally random first generation, the CEEFOOD will end with arbitrary designs that highly depend on early-emerged elites. If the user has some preference on some properties or features of the final design, it is always better to frame out and implement an initial guess than to let the CEEFOOD to search randomly.

Aside from the fundamental study on the design methods, I applied them on two open origami design problems, including the design of a flat-foldable shape with a desired folded profile area and the design of a rigid-foldable water container with a desired volume. The applications have proved that the GA with a proper geometric representation for origami structure can be a great tool for searching solutions for NP-complete origami design problems.

The primary advantage of the GA-based meta-heuristics on the open origami design problem is their capability of realizing the design objectives that are not achievable by existing origami design methods, such as "making the flat-folded shape's center of mass overlap with one of its creases". But the proposed GA-based meta-heuristics are also great tools for exploiting new and novel features that are not initially expected by the designers or the design problems. And if these new features fit for the design problem, the designers can then formulate the new features found into a new fitness evaluation component and update the design fitness function.

Future Work

There are two major directions for planning future studies. One direction is about the mathematics, geometry and algorithms in origami. This direction can be specified into the following specifications:

1) The inspection of flat-foldability and rigid-foldability of an origami is left partially solved. Although both of them are NP-complete problems, the research can proceed on the improvement of the current algorithms.

2) The current algorithm for extracting the creases from a PMR uses density-based clustering and LDA. Future study can be invested in the two algorithms to improve the efficiency of the entire CRA.

The other direction is about the application of the GA on more types of open origami design problems.

1) In this dissertation, the GA and its variation have proved their effectiveness on open origami design problems requesting both geometric and functional problems. But the applications of the method that has been presented so far only discussed about realizing functional requirements heavily relying on geometry. In future study, I plan to expand the method by achieving the realization of functional requirements related with structural properties. For instance, one of the problems I am interested in is to synthesize the FEA of the structure into both the GA with "ice-cracking" and the CEEFOOD. So the methods would be available for designing origami shapes and their material that have enough strength for supporting a load. Another extension would be targeted at some practical origami design problems, such as an origami tent with desired space inside, and a paper plane with desired aerodynamic properties.
2) The research has been focusing on the mathematic and geometric foundations of the abductive design method. The future study is on extending the method on practical engineering problems. One possible application is realizing crease pattern designs on an intelligent SMA origami sheet based on the research of the Texas A&M University ODISSEI group (Hartl, Lane and Malak 2012, Oehler,

Hartl, Lopez, Malak et al. 2012, Peraza-Hernandez, Hartl and Malak 2013, Peraza-Hernandez, Hartl and Malak 2013). The SMA origami sheet features the capability of bending along arbitrary creases while keeping the original shape of the sheet unchanged. But the SMA origami sheet won't allow sharp folding angles, it is more suitable to be used for making foldable structures to achieve static or dynamic functional properties than to be used for building exact artistic shape designs. Therefore, after interfacing with structural analysis tools, the method developed in this research will be a great tool for deriving shape designs for the SMA origami sheet.

3) The complementary mechanisms (captive breeding and atavism) and the hybrid ranking are applied to support the basic evolutionary operators in keeping the balance between elitism and diversity among the candidate solutions through the evolution of the GA and CEE. But I plan to make the statistical analysis of how these measures, as well as the parameter settings for these evolutionary operators and mechanisms, would affect the convergence speed and the quality of designs.

REFERENCES

Amato, N. M., Dill, K. A. and Song, G. (2003). "Using Motion Planning to Map Protein Folding Landscapes and Analyze Folding Kinetics of Known Native Structures." Journal of Computational Biology 10(3-4): 239-255.

Amato, N. M. and Song, G. (2002). "Using Motion Planning to Study Protein Folding Pathways." Journal of Computational Biology 9(2): 149-168.

Balkcom, D. J., Demaine, E. D. and Demaine, M. L. (2004). "Folding Paper Shopping Bags." 14th Annual Fall Workshop on Computational Geometry, MIT, Cambridge, MA.

Balkcom, D. J. and Mason, M. T. (2008). "Robotic Origami Folding." The International Journal of Robotics Research 27(5): 613-630.

Bateman, A. (2001). "Computer Tools and Algorithms for Origami Tessellation Design." The 3rd International Meeting of Origami Science, Math, and Education, Pacific Grove, CA.

Bentley, P. and Kumar, S. (1999). "Three Ways to Grow Designs: A Comparison of Evolved Embryogenies for a Design Problem." Genetic and Evolutionary Computation Conference, Orlando, FL.

Bern, M. and Hayes, B. (1996). "The Complexity of Flat Origami." 7th Annual ACM-SIAM Symposium on Discrete Algorithms, Atlanta, GA.

Bollinger, K., Grohmann, M. and Tessmann, O. (2010). "Structured Becoming: Evolutionary Processes in Design Engineering." Archetectural Design 1(206): 34-39.

Bowers, C. P. (2005). "Simulating Evolution with a Computational Model of Embryogeny: Obtaining Robustness from Evolved Individuals." In Advances in Artificial Life, Proceeding of the 8th European Conference on Artificial Life, Canterbury, United Kingdom.

Bowers, C. P. (2008). "Modularity in a Computational Model of Embryogeny." Design by Evolution. 3: 243-263.

Coello, C. A. C. (1999). "A Comprehensive Survey of Evolutionary Algorithm: Objective Optimization Techniques." Knowledge Information Systems 1(3): 269-308.

Coello, C. A. C. (2009). "Evolutionary Multi-objective Optimization: Some Current Research Trends and Topics that Remain to be Explored." Frontiers of Computer Science in China 3(1): 18-30.

Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2002). "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II." IEEE Transactions on Evolutionary Computation 6(2): 182-197.

Deb, K. and Srinivasan, A. (2006). "Innovization: Innovating Design Principles through Optimization." Genetic and Evolutionary Computation Conference, Seattle, WA.

Deb, K. and Tiwari, S. (2005). "Multi-objective Optimization of a Leg Mechanism Using Genetic Algorithms." Engineering Optimization 37(4): 325-350.

Demaine, E. D. and Demaine, M. L. (2001). "Recent Results in Computational Origami." The 3rd International Meeting of Origami Science, Math, and Education, Pacific Grove, CA.

Demaine, E. D., Demaine, M. L. and Ku, J. (2011). "Folding Any Orthogonal Maze." Fifth International Meeting of Origami Science, Mathematics, and Education, Singapore.

Demaine, E. D. and O'Rourke, J. (2007). "Geometric Folding Algorithms: Linkages, Origami, Polyhedra." Cambridge, United Kingdom, Cambridge University Press.

El-Sonbaty, Y., Ismail, M. and Farouk, M. (2004). "An Efficient Density Based Clustering Algorithm for Large Databases." 16th IEEE International Conference on Tools with Artificial Intelligence, Boca Raton, FL.

Esquivel, G., Xing, Q., Collier, R., Tomaso, M. and Akleman, E. (2011). "Weaving Methods in Architectural Design." Bridges 2011 (Connections between mathematics, art, and scsience), Combria, Portugal.

Ester, M., Kriegel, H., Sander, J. and Xu, X. (1996). "A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR.

Ester, M., Kriegel, H., Sander, J. and Xu, X. (1997). "Density-connected Sets and Their Application for Trend Detection in Spatial Databases." In Third International Conference on Knowledge Discovery and Data Mining, Menlo Park, CA.

Farnsworth, M., Benkhelifa, E., Tiwari, A. and Zhu, M. (2010). "A Novel Approach to Multi-level Evolutionary Design Optimization of a MEMS Device." Lecture Notes in Computer Science 6274: 322-334.

Fastag, J. (2006). "eGame: Virtual Paperfolding and Diagramming Software." Fourth International Meeting of Origami Science, Mathematics, and Education, Pasadena, CA.

Fieldsend, J. E., Everson, R. M. and Singh, S. (2003). "Using Unconstrained Elite Archives for Multiobjective Optimization." IEEE Transactions on Evolutionary Computation 7(3): 305-323.

Fonseca, C. M. and Fleming, P. J. (1993). "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization." Genetic Algorithms: Proceedings of the Fifth International Conference, San Mateo, CA.

Fonseca, C. M. and Fleming, P. J. (1998). "Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms. I. A Unified Formulation." Systems, Man and Cybernetics, Part A: Systems and Humans 28(1): 26-37.

Fuse, T. (1990). "Unit Origami: Multidimensional Transformations." Tokyo, Japan, Japan Publications : U.S. Distributor Kodansha America through Farrar, Straus & Giroux.

Gantes, C. J. (2001). "Deployable Structures: Analysis and Design." Ashurst, United Kingdom, WIT Press.

Garey, M. R. and Johnson, D. S. (1979). "Computers and Intractability: A Guide to the Theory of NP-Completeness." London, United Kingdom, W. H. Freeman.

Gen, M. and Cheng, R. (1999). "Genetic Algorithms and Engineering Optimization." Hoboken, NJ, Wiley-Interscience.

Goldberg, D. E. (1989). "Genetic Algorithms in Search, Optimization, and Machine Learning." Boston, MA, Addison-Wesley Professional.

Goldberg, D. E. and Richardson, J. (1987). "Genetic Algorithms with Sharing for Multimodal Function Optimization." Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and Their Application, Cambridge, MA.

Harbin, R. (1997). "Secrets of Origami: the Japanese Art of Paper Folding." Mineola, NY, Courier Dover Publications.

Hartl, D., Lane, K. and Malak, R. (2012). "Computational Design of a Reconfigurable Origami Space Structure Incorporating Shape Memory Alloy Thin Films." ASME 2012 Conference on Smart Materials, Adaptive Structures and Intelligent Systems, Snowbird, UT.

Hawkes, E., An, B., Benbernou, N. M., Tanaka, H., Kim, S., Demaine, E. D., Rus, D. and Wood, R. J. (2010). "Programmable Matter by Folding." Proceedings of the National Acadamies of Science 107(28): 12441-12445.

Huffman, D. A. (1976). "Curvature and Creases: A Primer on Paper." IEEE Transactions on Computers 25(10): 1010-1019.

Hull, T. (1994). "On the Mathematics of Flat Origamis." Congressus Numerantium 100: 215-224.

Hull, T. (2006). "Project Origami: Activities for Exploring Mathematics." Boca Raton, FL, A.K. Peters/CRC Press.

Huzita, H. and Mitchell, M. (1989). "The Algebra of Paper-folding." First International Meeting of Origami Science and Technology, Ferrara, Italy.

Jensenm, M. T. (2003). "Reducing the Run-Time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms." IEEE Transactionson Evolutionary Computation 7: 503-515.

Kicinger, R., Arciszewski, T. and De Jong, K. (2005). "Evolutionary Computation and Structural Design: A Survey of the State-of-the-art." Computers & Structures 83(23-24): 1943-1978.

Knowles, J. and Corne, D. (1999). "The Pareto Archived Evolution Strategy : A New Baseline Algorithm for Pareto Multiobjective Optimisation " CEC 99. Proceedings of the 1999 Congress on Evolutionary Computation, Washington, DC.

Konak, A., Coit, D. W. and Smith, A. E. (2006). "Multi-objective Optimization Using Genetic Algorithms: A Tutorial." Reliability Engineering and System Safety 91(9): 992-1007.

Konak, A. and Smith, A. E. (2002). "Multiobjective Optimization of Survivable Networks Considering Reliability." Proceedings of the 10th International Conference on Telecommunication Systems, Monterey, CA.

Konjevod, G. (2006). "Integer Programming Models for Flat Origami." Fourth International Meeting of Origami Science, Mathematics, and Education, Pasadena, CA.

Lang, R. J. (1996). "A Computational Algorithm for Origami Design." 12th Annual ACM Symposium on Computational Geometry, Philadelphia, PA.

Lang, R. J. (2003). "Origami Design Secrets: Mathematical Methods for an Ancient Art." Boca Raton, FL, A. K. Peters/CRS Press.

Lang, R. J. (2005). "Origami Design Secrets: Mathematical Methods for an Ancient Art." Mathematics And Statistics, The Mathematical Intelligencer 27(2): 92-95.

Lei, X. and Shi, Z. (2004). "Overview of Multi-objective Optimization Methods." Journal of Systems Engineering and Electronics 15(2): 142-146.

Li, W. and McAdams, D. A. (2013). "A Novel Pixelated Multicellular Representation for Origami Structures That Innovates Computational Design and Control." ASME 2013 International Design Engineering Technical Conferences (IDETC) and Computers and Information in Engineering Conference (CIE), Portland, OR.

Lin, L., Gen, M. and Wang, X. (2009). "Integrated Multistage Logistics Network Design by Using Hybrid Evolutionary Algorithm." Computers and Industrial Engineering 56(3): 854-873.

Mandal, D., Bhattacharjee, A. K. and Ghoshal, S. P. (2009). "Comparative Optimal Designs of Non-uniformly Excited Concentric Circular Antenna Array Using Evolutionary Optimization Techniques." 2nd International Conference on Emerging Trends in Engineering and Technology, London, United Kingdom.

MathWorks. (2014). "Digital Modulation." MatLab Documentation Center, 2014, from http://www.mathworks.com/help/comm/ug/digital-modulation.html.

Mitani, J. (2011). "A Method for Designing Crease Patterns for Flat-Foldable Origami with Numerical Optimization." Journal for Geometry and Graphics 15(2): 195-201.

Miura, K. (2006). "The Science of Miura-Ori: A Review." Fourth International Meeting of Origami Science, Mathematics, and Education, Pasadena, CA.

Miura, K. (2009). "Triangles and Quadrangles in Space." the International Association for Shell and Spatial Structures (IASS) Valencia, Spain.

Montroll, J. (1979). "Origami for the Enthusiast: Step-By-Step Instructions in over 700 Diagrams." Mineola, NY, Courier Dover Publications.

Nagpal, R. (2002). "Programmable Self-Assembly Using Biologically-Inspired Multiagent Control." 1st International Joint Conference on Autonomous Agents and Multiagent Systems, Bologna, Italy.

Nagpal, R., Kondacs, A. and Chang, C. (2003). "Programming Methodology for Biologically-Inspired Self-Assembling Systems." AAAI Spring Symposium on Computational Synthesis, Menlo Park, CA.

O'Neill, M., McDermott, J., Mark Swafford, J., Byrne, J., Hemberg, E., Brabazon, A., Shotton, E., McNally, C. and Hemberg, M. (2010). "Evolutionary Design Using Grammatical Evolution and Shape Grammars: Designing a Shelter." International Journal of Design Engineering 3(1): 4-24.

Oehler, S., Hartl, D., Lopez, R., Malak, R. and Lagoudas, D. (2012). "Design Optimization and Uncertainty Analysis of SMA Morphing Structures." Smart Materials and Structures 21(9): 1-16.

Oezluek, A. C., Dibowski, H. and Kabitzsch, K. (2009). "Automated Design of Room Automation Systems by Using an Evolutionary Optimization Method." IEEE Conference on Emerging Technologies and Factory Automation, Mallorca, Spain.

Ovadya, A. (2010). "Origami Transformers: Folding Orthogonal Structures from Universal Hinge Patterns." M. Eng, Massachusetts Institute of Technology.

Palmer, C. C. and Kershenbaum, A. (1994). "Representing Trees in Genetic Algorithms." Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on, Orlando, FL.

Peraza-Hernandez, E., Hartl, D. and Malak, R. (2013). "Simulation-based Design of a Self-folding Smart Material System." ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Portland, OR.

Peraza-Hernandez, E. A., Hartl, D. J. and Malak, R. J. (2013). "Design and Numerical Analysis of an SMA Mesh-based Self-folding Sheet." Smart Materials and Structures 22.

Pettersson, F., Saxen, H. and Deb, K. (2009). "Genetic Algorithm-Based Multicriteria Optimization of Ironmaking in the Blast Furnace." Materials And Manufacturing Processes 24(3): 343-349.

Poma, F. (2009). "On the Flat-Foldability of a Crease Pattern." from http://poisson.phc.unipi.it/~poma/Ffcp.pdf.

Potter, M. A. and De Jong, K. A. (1994). "A Cooperative Coevolutionary Approach to Function Optimization." Parallel Problem Solving from Nature - PPSN III: 249-257.

Potter, M. A. and De Jong, K. A. (2000). "Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents." Evolutionary Computation 8(1): 1-29.

Reddy, J. N. (2005). "An Introduction to the Finite Element Method." New York City, McGraw-Hill Science/Engineering/Math.

Schmitt, L. M. (2001). "Theory of Genetic Algorithms." Theoretical Computer Science 259: 1-61.

Schneider, J. (2004). "Flat-Foldability of Origami Crease Patterns." from http://www.sccs.swarthmore.edu/users/05/jschnei3/origami.pdf.

Sharma, D., Deb, K. and Kishore, N. N. (2013). "Customized Evolutionary Optimization Procedure for Generating Minimum Weight Compliant Mechanisms." Engineering Optimization: 39-60.

Stoy, K. (2004). "Self-Repair Through Scale Independent Self-Reconfiguration." Proceedings of IEEE/RSJ International Conference on Robots and Systems, Sendai, Japan.

Stroble, J. K., Nagel, R. L., Stone, R. B. and McAdams, D. A. (2010). "Function-Based Biology Inspired Concept Generation." Artificial Intelligence for Engineering Design, Analysis and Manufacturing 24(4): 521-535.

Tachi, T. (2006). "3D Origami Design based on Tucking Molecule." Fourth International Meeting of Origami Science, Mathematics, and Education, Pasadena, CA.

Tachi, T. (2006). "Simulation of Rigid Origami." Fourth International Meeting of Origami Science, Mathematics, and Education, Pasadena, CA.

Tachi, T. (2009). "Generalization of Rigid Foldable Quadrilateral Mesh Origami." the International Association for Shell and Spatial Structures (IASS), Valencia, Spain.

Tachi, T. (2010). "Origamizing Polyhedral Surfaces." IEEE Transactions on Visualization and Computer Graphics 16(2): 298-311.

Tachi, T. (2010). "Rigid-foldable Thick Origami." Fifth International Meeting of Origami Science, Mathematics, and Education, Singapore.

Tibbits, S. (2012). "Design to Self-Assembly." Architectural Design 82(2): 68-73.

Tibbits, S. and Cheung, K. (2012). "Programmable Materials for Architectural Assembly and Automation." Assembly Automation 32(3): 216-225.

Trudeau, R. J. (1994). "Introduction to Graph Theory." Mineola, NY, Courier Dover Publications.

Van Den Berg, J., Miller, S., Goldberg, K. and Abbeel, P. (2011). "Gravity-Based Robotic Cloth Folding." Algorithmic Foundations of Robotics IX. Berlin, Germany, Springer Berlin Heidelberg: 409-424.

Wang, K. and Chen, Y. (2010). "Folding a Patterned Cylinder by Rigid Origami." Fifth International Meeting of Origami Science, Mathematics, and Education, Singapore.

Watanabe, N. and Kawaguchi, K.-i. (2006). "The Method for Judging Rigid Foldability." Fourth International Meeting of Origami Science, Mathematics, and Education, Pasadena, CA.

Wikipedia. (2014). "History of Origami." from http://en.wikipedia.org/wiki/History_of_origami.

Yogev, O., Shapiro, A. A. and Antonsson, E. K. (2010). "Computational Evolutionary Embryogeny." Evolutionary Computation 14(2): 301-325.

Yukokoji, Y., Tanaka, K. and Kamotani, Y. (2006). "Origami Folding by a Robotic Hand." IEIC Technical Report (Institute of Electronics, Information and Communication Engineers) 106: 113-118.

Zitzler, E. and Thiele, L. (1999). "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach." IEEE Transactions on Evolutionary Computation 3(4): 257-271.