A Novel Method to Design and Optimize Flat-Foldable Origami Structures Through a Genetic Algorithm

Daniel A. McAdams[1]
Department of Mechanical Engineering,
Texas A&M University,
3123 TAMU,
College Station, TX 77843-3123
e-mail: dmcadams@tamu.edu

Wei Li
Department of Mechanical Engineering,
Texas A&M University,
3123 TAMU,
College Station, TX 77843-3123
e-mail: liwei.meen@gmail.com

As advantages of foldable or deployable structures have been established, origami artists and engineers have started to study the engineering applications of origami structures. Methods of computational origami design that serve different types of origami have already been developed. However, most of the existing design methods focus on automatically deriving the crease pattern to realize a given folded finished shape, without actually designing the finished shape itself. To include final shape design into the computational origami design and optimization process, this paper presents a genetic algorithm that aims to develop origami structures featuring optimal geometric, functional, and foldability properties. In accordance with origami, the genetic algorithm is adapted both in the aspects the individual encoding method and the evolutionary operators. To compliment the Genetic algorithms (GA), a new origami crease pattern representation scheme is created. The crease pattern is analogous to the ice-cracks on a frozen lake surface, where each crack is equivalent to a crease and each forking point to a vertex. Thus to form the creases and vertices in an "ice-cracking"-like origami crease pattern, we pick one vertex as the starting location, and let the rest of the creases and vertices grow in the same manner that cracks extend and fork form in ice. In this research, the GA encodes the geometric information of forming the creases and vertices according to the development sequence through the ice-cracking process. Meanwhile, we adapt the evolutionary operators and introduce auxiliary mechanisms for the GA, so as to balance the preservation of both elitism and diversity and accelerate the emergence of optimal design outcomes through the evolutionary design process. [DOI: 10.1115/1.4026509]

1 Introduction

Origami is traditionally a manual form of art that realizes 3D shapes by folding paper sheets [1]. Recently, researchers have discovered origami applications engineering [2]. Thus, the field of origami engineering is growing both with the development of mathematical and computational methods for designing folded shapes, and the applications of folded origami style geometries in engineering applications. Several computational origami design methods [3–9] have been developed. These methods, such as TreeMethod [6] and Origamizer [8,9], generally focus on deriving a crease pattern for a specific and known final folded origami shape. The folded shape is treated as an input or a constraint to the crease design problem, but is not considered as part of computational design problem.

In this study, we develop a method to allow the final folded shape to be part of the determined design. Without a known folded origami shape, the topological structure of the corresponding crease pattern is hard to predict. Thus to derive a viable solution, a design realization method has to explore a large and non-linear design search space, which covers all possible formations of vertices and creases. Genetic algorithms (GA) [10] have proven successful as a method for approaching challenging non-linear optimization problems [11]. Thus, we choose to use GA methods for the origami design problem. In order to apply GAs to the origami crease and shape design problems, there are two key problems that must be solved. The first problem deals with how to use a binary or Gray code string to encode any candidate crease pattern solution and the creation of some compatible crease representation scheme, while the second consists of developing a method to accelerate and regulate the evolution process.

Flat-foldability is a constraint often applied to origami crease patterns. In practice, origamis are commonly flat-foldable as flat-foldability is satisfied by most traditional origami art made through manual folding and unfolding only [12]. Flat-foldability indicates that an origami structure can be flattened without being crumpled, bent, or damaged. There are two types of flat-foldability: local flat-foldability and global flat-foldability. Local flat-foldability only concerns whether the immediate region around each internal vertex can be flat-folded. The crease pattern of a vertex's local region is locally flat-foldable, as long as it satisfies the Maekawa–Justin theorem, the Kawasaki–Justin theorem, and the local min theorem [13]. On the other hand, global flat-foldability considers the full crease pattern. A globally flat-foldable origami must be locally flat-foldable everywhere. In addition, the globally flat-foldable origami should satisfy the condition of having no collisions of the sheet with itself during the folding procedure. Previously, the authors have provided a heuristic for the flat-foldability identification of a known crease arrangement [14]. The basic idea is to tree-search all the Mountain Valley (MV)-assignments and face overlapping orders, while immediately pruning the "branches" of solutions as they violate local flat-foldability [4] or the non-collision criteria [15,16]. Although the tree-search heuristic is a "brute force" solution, it is still a serviceable solution to the flat-foldability identification problem for origami with a limited number of creases (NOC).

In this paper, we present a new encoding method suitable for both general origami and flat-foldable origami. The key contribution of the developed method is that it naturally combines a geometric representation for creases and the GA encoding requirements. The method encodes the vertices and creases in an origami

[1]Corresponding author.

Contributed by the Design Engineering Division of ASME for publication in the Journal of Computing and Information Science in Engineering. Manuscript received October 23, 2013; final manuscript received January 15, 2014; published online June 2, 2014. Editor: Bahram Ravani.

according to an ice-cracking sequence, as we take the cracks on an icy surface to be a structural analogy to the crease pattern. Section 2 presents the basic method and representation. Section 3 provides the specific GA adaptations we have made for the origami design problem here. These adaptations include consideration of elitism and diversity preservation, as well as the application of multi-objective Pareto ranking among individuals. In Sec. 4, we provide an example design created through the adapted GA for an origami design and optimization problem.

2 Encoding Origami Structures by Ice-Cracking Sequence

The essential problem of setting up a GA encoding method for our crease representation is to determine in which order the pieces of code for vertices and creases are arranged in the complete genetic code sequence. Note that a crease pattern can be defined as a network or networks of vertices and creases, whereas a crease connects two vertices, and a vertex emits several creases.

2.1 Embedded Geometric Representation for Crease Patterns. A typical embedded geometric representation for crease patterns is to list the creases by their end points. For instance, the crease pattern shown in Fig. 1(*a*) has four creases, which 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 genetic code can be derived by transforming the argument set A_1 to Gray code. The genetic code of the crease pattern based on the argument set A_1 is

$$\begin{aligned} C_1 = {}&''0111-0101-0000-0110-0111-0101-0011-0000\\ &-0111-0101-1111-0010-0111-0101-0110\\ &-1111'' \end{aligned}$$

This embedded representation will result in very "fragile" genetic codes. Here, "fragile" means that any single bit of mutation in the genetic code will frequently cause the corresponding crease pattern to change significantly and become invalid. For example, consider the crease pattern in Fig. 1(*a*). When its genetic code 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 Fig. 1(*b*). In Fig. 1(*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 detached from them.

To avoid the mutation of the genetic code resulting in the detachment of connected creases, an alternative geometric representation, which is abstracted from Mitani's method of generating random crease patterns [12], uses another set of arguments (A_2) to define the crease pattern in Fig. 1(*a*):

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

In A_2, the starting five pairs of arguments list the locations of the five vertices that are implicitly numbered as #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. The genetic code for A_2 is

$$\begin{aligned} C_2 = {}&''0111-0111-0101-0000-0110-0011-0000\\ &-1111-0010-0110-1111-001-011-001-010\\ &-001-110-001-111'' \end{aligned}$$

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, so that a single mutation won't result in an invalid crease pattern. However, the mutation of the bits that define the creases will cause the loss of some creases and result in an invalid crease pattern. For example, mutation of the last bit of C_2 will change the last argument in A_2 from 5 to 4. Therefore, the corresponding crease pattern (Fig. 1(*c*)) of the mutated C_2 has two overlapping creases between 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 counted as one, the crease pattern (Fig. 1(*c*)) is not foldable, as it has a vertex (#1) that is connected with three creases.

A common problem of the embedded and the alternative geometric representations is that arbitrary genetic codes do not always define a valid crease pattern. Thus, neither of the two is well suited as an encoding method in GA. Therefore, it is necessary to create a new geometric representation, so that any random genetic code can define a valid crease pattern, and any valid crease pattern doesn't become invalid as its genetic code mutates. Moreover, the new representation will also include a solution to guarantee the crease patterns to be flat-foldable (as explained in Sec. 2.3).

2.2 Defining Vertices and Creases by the Ice-Cracking Sequence. In this research, we use the growth of a crack on an icy surface as an analogy for a representation of an origami crease

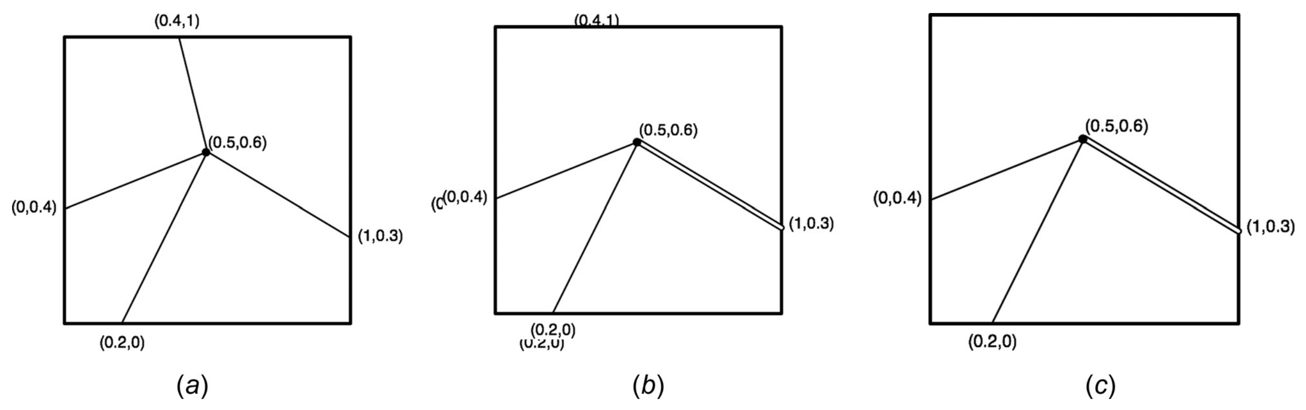

Fig. 1 **(*a*) A crease pattern represented by the locations of the vertices and the creases that link in between; (*b*) the mutation on one bit in the genetic code for the first intuitive geometric representation of (*a*) results in an invalid crease pattern with a breakage; (*c*) the mutation on 1 bit in the genetic code for the second intuitive geometric representation of (*a*) results in an invalid crease pattern due to the missing of one crease**

pattern. With respect to the ice-cracks, creases are equivalent to cracks, and vertices are equivalent to forkings. Thus, a crease pattern in an origami sheet can be formed in a similar manner as the growth of ice-cracks and forks on the icy surface. We name this approach to growing vertices and creases ice-cracking. If the vertices and creases are generated only one at a time, a sequence of vertices and creases being generated is also attained as a by-product of the ice-cracking.

To form a pinwheel crease pattern as shown in Fig. 2(*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 NOC is four according to Fig. 2(*a*). Four direction vectors then emanate from the #1 vertex. Here, we name the direction vectors *dummy creases*, 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 Fig. 2(*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 Figs. 2(*a*) and 2(*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, we 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 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 Fig. 2(*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.

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 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 Fig. 2(*d*).

In Fig. 2(*d*), #2 and #10 dummy creases have an intersection. In this step, we 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 NOC of the #4 vertex is four.

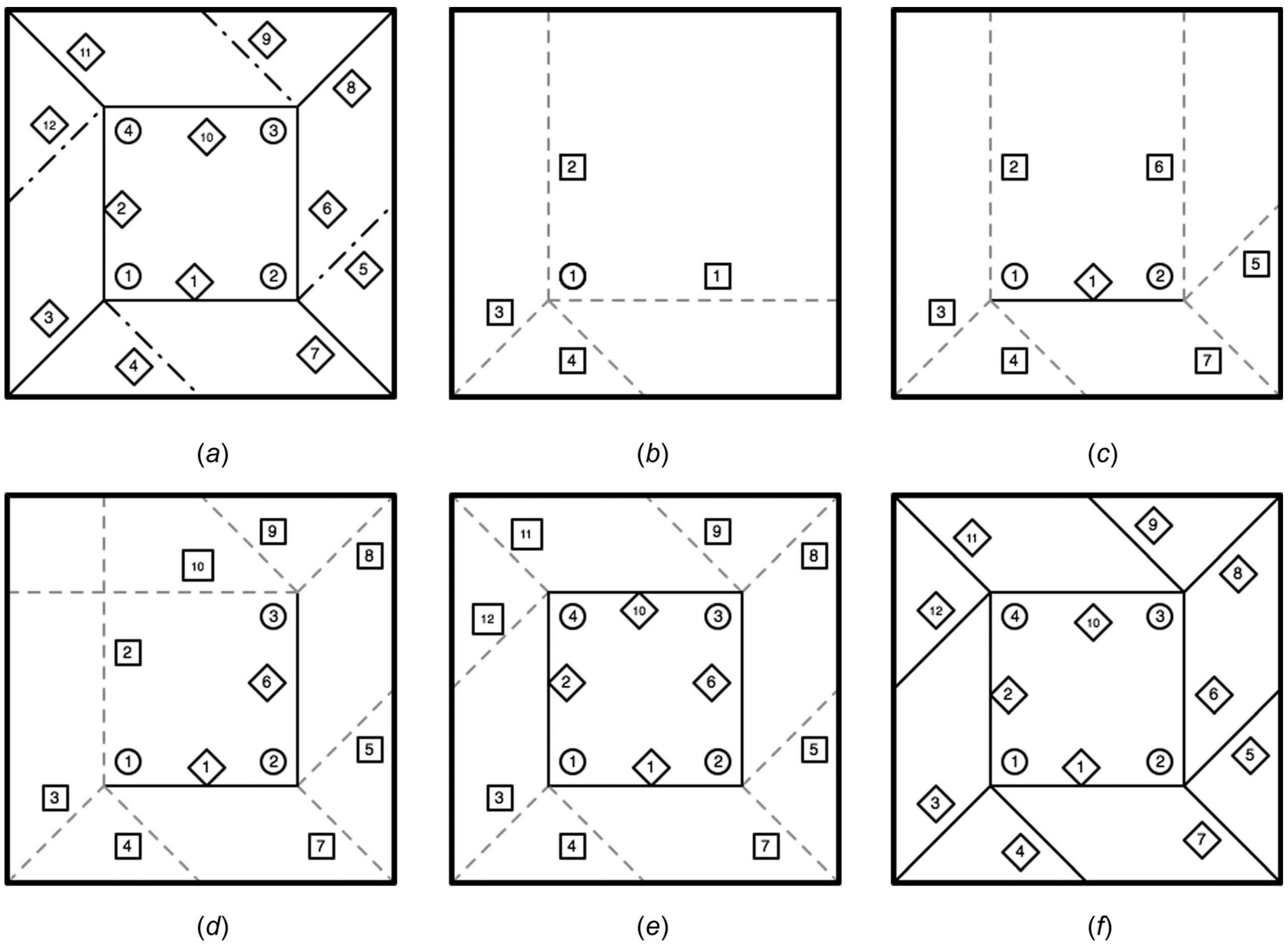

Fig. 2 The steps taken by ice-cracking to derive the vertices and creases through a systematic sequence. (*a*) The full crease pattern with circled numbers labeling the vertices and diamonded numbers labeling the creases; (*b*) the initialization step that locates the first vertex; (*c*) the first forking 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.

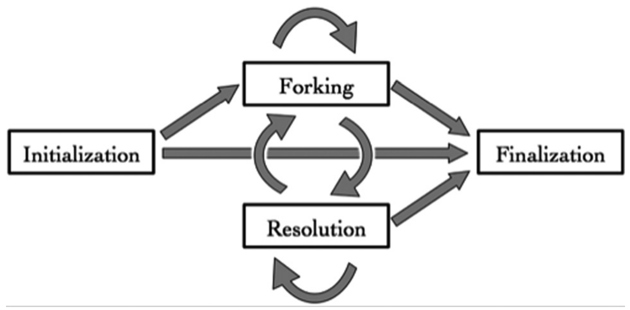

Fig. 3 The four steps—initialization, forking, resolution and finalization—of ice-cracking. The arrows show how the different steps can be arranged, with initialization being the first step and finalization the last step.

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 Fig. 2(*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 Fig. 2, the finalization step will derive all the creases of Fig. 2(*f*) without MV-assignment.

To determine the MV-assignment of the crease arrangement as shown in Fig. 2(*f*), we need to first clarify the foldability requirements. As for the flat-foldability, we are able to use the method given in Ref. [14]. 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.

2.3 Summary of Ice-Cracking and Its Encoding Method.

The example in Sec. 2.3 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 Fig. 3. 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.

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 then encode the operations of the step in a fixed format.

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 binary genetic code is presented as well. In this paper, the conversion from binary to Gray code uses the M-QAM modulation, which is different from the natural Gray code translation [17].

2.3.1 *Initialization Step*.

Using ice-cracking, any crease pattern must have one and only one initialization step. Two main operations are done 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 NOC and the angles of the dummy creases follow the allocation of the vertex. In general, if the first vertex has a 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 Fig. 2(*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 pattern to be 256×256 ($2^8 \times 2^8$), 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 NOC, usually defined as a range. For instance, we could have a range of $4 \le \text{NOC} \le 11$. In this case, 3 bits are needed to define the NOC among the eight choices. However, if the crease pattern is constrained to be flat-foldable, the number of NOC for each vertex

Table 1 The operations in each step, and the arguments that define 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 NOC	NOC
	3. Provide the absolute angle of one dummy crease	θ
	4. Draw the other dummy creases	$\kappa_i (i = 1, \ldots, \text{NOC})$
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 NOC	NOC
	5. Draw the dummy creases	$\kappa_i (i = 1, \ldots, \text{NOC})$
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 NOC	NOC
	4. Draw the dummy creases	$\kappa_i (i = 1, \ldots, \text{NOC})$
Finalization step/11	1. Eliminate the intersections of dummy creases	Fixed implicit argument set $\{n_{\text{VI}} = 1, \text{NOC} = 4, \kappa_1 = 255, \kappa_2 = 255, \kappa_3 = 255\}$
	2. Convert dummy creases to creases	
	3. Terminate the ice-cracking	

must be even according to Maekawa–Justin Theorem [4]. Thus a flat-foldable vertex that picks its NOC among the span of {4, 6, 8, or 12} requires only 2 bits.

If the 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 with respect to the 0-rad line. Here, the #1 dummy crease becomes the baseline, and its absolute angle with respect to 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, since it provides an additional constraint argument 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

$$\sum_{i\in\text{odd}} \alpha_i = \sum_{i\in\text{even}} \alpha_i = \pi \tag{3.1}$$

Hence for Fig. 4(*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

$$\alpha_i = \begin{cases} \pi \times \dfrac{\kappa_i}{\sum\limits_{j\in\text{odd}} \kappa_j} & (i \text{ is odd}) \\ \pi \times \dfrac{\kappa_i}{\sum\limits_{j\in\text{even}} \kappa_j} & (i \text{ is even}) \end{cases} \tag{3.2}$$

In all, the entire initialization step for the flat-foldable #1 vertex of pinwheel pattern could use one possible argument set $\{x_1 = 0.25,\ y_1 = 0.25,\ \text{NOC} = 4,\ \theta = 0,\ \kappa_1 = 255,\ \kappa_2 = 255,\ \kappa_3 = 255,\ \kappa_4 = 85\}$, which is then encoded by: "00 01110000 01110000 00 00000000 10101010 10101010 10101010 01100110."

2.3.2 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 n_{DC} indicates that the new vertex is on the 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 n_{DC}. If n_{DC} exceeds the total number N_{DC} of the existing dummy creases, we instead use the remainder of n_{DC} divided by N_{DC}. The second argument λ, which also takes 8 bits in our 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

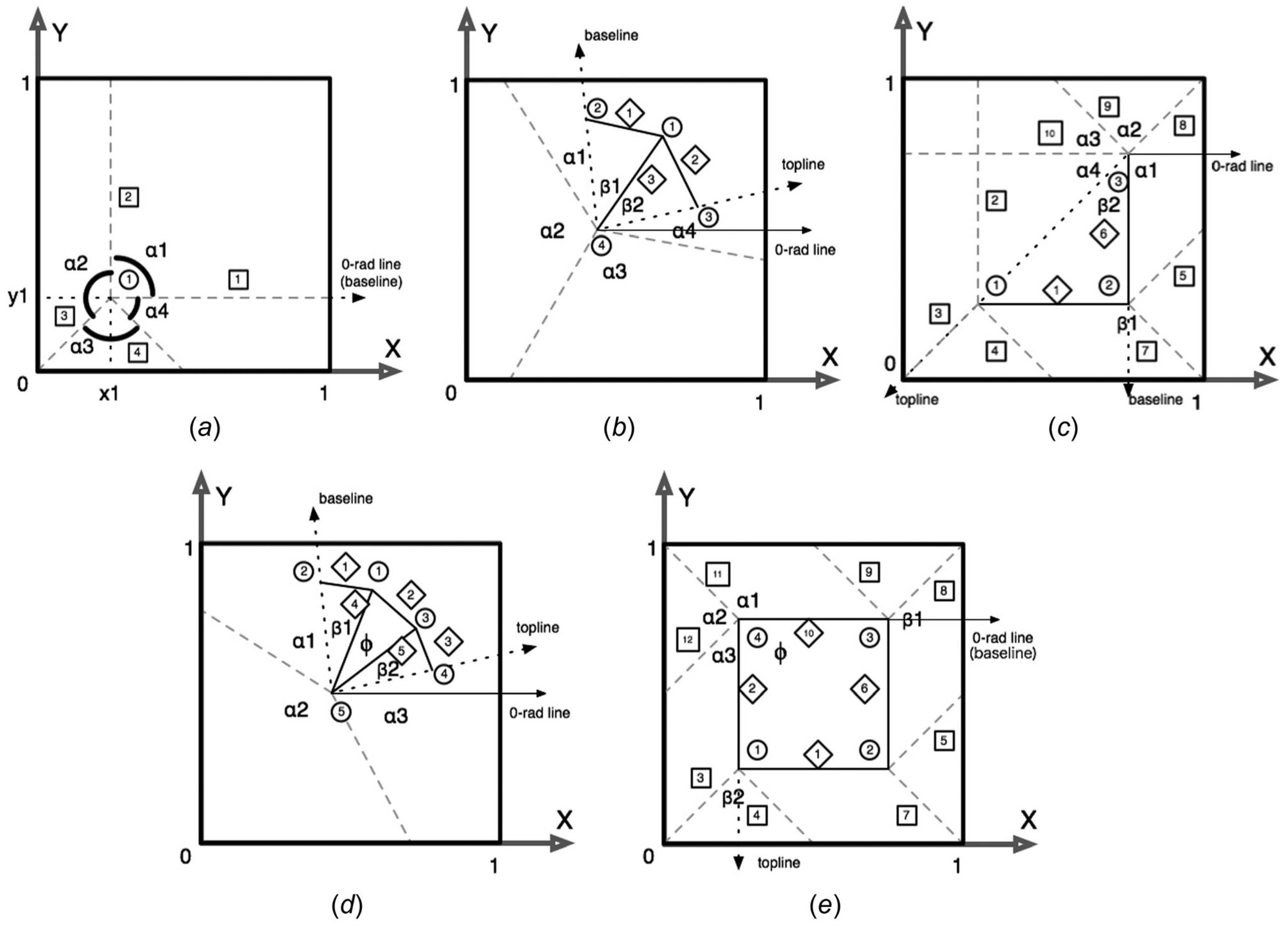

Fig. 4 Illustrative patterns. (*a*) Initialization step; (*b*) forking step; (*c*) the second forking step for getting a pinwheel pattern; (*d*) resolution step; (*e*) the resolution step for getting the #4 vertex in a pinwheel pattern.

$$d = l \times \frac{\lambda}{2^8} \tag{3.3}$$

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 NOC of the new vertex in the same way as in the initialization step. Suppose that in Fig. 4(*b*) the #4 vertex is the new vertex determined in a prior forking step. If we already know that the 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 was introduced by the foldability requirement.

For a flat-foldable crease pattern, 4 arguments are required. As shown in Fig. 4(*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 β_1 as the angle between the #3 crease and the baseline, while β_2 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 α_1 to α_4 represent the angles among the baseline, topline, and new dummy creases as shown in Fig. 4(*b*). According to Kawasaki–Justin theorem, we have

$$\beta_1 + \sum_{i \in \text{odd}} \alpha_i = \beta_2 + \sum_{i \in \text{even}} \alpha_i = \pi \tag{3.4}$$

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

$$\alpha_i = \begin{cases} (\pi - \beta_1) \times \dfrac{\kappa_i}{\sum_{j \in \text{odd}} \kappa_j} & (i \text{ is odd}) \\ (\pi - \beta_2) \times \dfrac{\kappa_i}{\sum_{j \in \text{even}} \kappa_j} & (i \text{ is even}) \end{cases} \tag{3.5}$$

In the same way, if we consider the derivation of the #3 vertex in the pinwheel pattern (Fig. 4(*c*)), the whole forking step could use one possible arguments set of $\{n_{\text{DC}} = 6,\ \lambda = 0.6667,\ \text{NOC} = 4,\ \kappa_1 = 255,\ \kappa_2 = 254,\ \kappa_3 = 85,\ \kappa_4 = 127\}$, and the Gray code thus is: "01 00000100 01100110 00 10101010 10101011 01100110 01011010."

2.3.3 Resolution Step. The resolution step is similar to the forking step except for the new vertex location. In the resolution step, the new vertex must be created at an intersection between $n_\text{I} (\geq 2)$ existing dummy creases. The first argument n_{VI} indicates which intersection is used. One implicit operation is 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 n_I intersecting dummy creases, this intersection is the nearest one from the corresponding origin vertex. To determine the argument n_{DC} in forking step, we pick the n_{VI}-th valid intersection for the new vertex. Similarly, if n_{VI} is larger than the total number N_{VI} of valid intersections, we use the remainder of n_{VI} divided by N_{VI}. Right after the allocation of the new vertex, n_I new creases can also be fixed in this resolution step.

The second argument for resolution step is the NOC. In the example case of Fig. 4(*d*), the #5 vertex is the new vertex based on creases #4 and #5. If we set the 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 β_1 (the angle between #4 crease and the baseline), β_2 (the angle between #5 crease and the topline), and ϕ (the angle between #4 crease and #5 crease). Then we let α_1 to α_3 represent the angles between the baseline, topline, and new dummy creases as shown in Fig. 4(*d*). According to the Kawasaki–Justin theorem, we have

$$\beta_1 + \beta_2 + \sum_{i \in \text{odd}} \alpha_i = \phi + \sum_{i \in \text{even}} \alpha_i = \pi \tag{3.6}$$

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

$$\alpha_i = \begin{cases} (\pi - \beta_1 - \beta_2) \times \dfrac{\kappa_i}{\sum_{j \in \text{odd}} \kappa_j} & (i \text{ is odd}) \\ (\pi - \phi) \times \dfrac{\kappa_i}{\sum_{j \in \text{even}} \kappa_j} & (i \text{ is even}) \end{cases} \tag{3.7}$$

In the same way, if we consider the derivation of the #4 vertex of the pinwheel pattern as in Fig. 4(*e*), the whole resolution step could use one possible arguments set of $\{n_{\text{VI}} = 1,\ \text{NOC} = 4,\ \kappa_1 = 255,\ \kappa_2 = 255,\ \kappa_3 = 85\}$, and the code will be: "10 00 10101010 10101010 01100110."

In some situations, 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 NOC-n_I, but not NOC-2.

2.3.4 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 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 $\{n_{\text{VI}} = 1,\ \text{NOC} = 4,\ \kappa_1 = 255,\ \kappa_2 = 255,\ \kappa_3 = 255\}$, until no more intersections exist. The argument set is a default setting. If there are some special design requirements, such as "the 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.

2.3.5 Summary. 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 the ice-cracking crease pattern representation scheme and its Gray encoding is 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 is still 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 NOCs—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 NOCs to change, the result may 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. Also, 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.

3 Genetic Algorithm

GA are search heuristics inspired by the process of natural evolution. They are 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 on both a well-defined design space as well as a design space that the designers do not intentionally seek.

The mapping between the ice-cracking code and the corresponding crease pattern allows the implementation of GA methods on origami crease pattern design problems. The ice-cracking basically provides a method to derive a genetic code for arbitrary or flat-foldable crease patterns, and it also guarantees that any Gray code string can be decoded to its unique crease pattern.

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. The 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.

3.1 Elitism. In this paper, 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.

In *captive breeding*, 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 Refs. [18–20]. 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 archive only stores 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, $n_b (< N_b)$ 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. 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.

3.2 Diversity. Although captive breeding and atavism work well to maintain the elite individuals in the population, they also cause severely pre mature convergence toward the early emerged elites. If the fitness evaluation is scalar, a higher mutation rate, 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. 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 paper is the use of a multi-objective fitness evaluation, as the origami design or self-folding control problems usually involves multiple objectives. 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 [21,22]. 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 are 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.

4 Experiment and Discussions

In this section, we introduce a simple origami design and optimization problem as described below.

Objective: Given a 1-by-1 square sheet with constant thickness and uniform mass density, design a flat-foldable crease pattern, so that after flat-folded, the profile has an area of approximately 0.25. Moreover, we assume that in the unfolded state, the origami sheet lies in a horizontal plane P_h; while in the flat-folded state, all the faces stand in a vertical plane P_v. During the folding procedure, one of the creases must be anchored steadily, so that the two planes P_h and P_v intersect at this anchored crease. The sheet should have a minimal vertical height displacement on the center of mass (CoM) between the unfolded and flat-folded states as described above.

Performance Measures:

1. The number of vertices shall not exceed 20, but must be larger than 1;
2. The number of creases shall not exceed 24;
3. The number of faces shall not exceed 14;
4. The degrees-of-freedom (DoF) shall not exceed 5, but a lower DoF is strongly preferred;
5. The creases must all be longer than 0.20.

A measure of achieving the design objective and performance measures will determine the calculation of a fitness value in GA. "The difference between flat-folded profile area and 0.25," "the change on the center of mass," "the difference between the DoF and 1," and "the difference between the shortest crease length and 0.20" are the four components of the fitness evaluation. As mentioned previously, both the fitness evaluation components of the multi-objective fitness vector and the normalized fitness value should fall between 0 and 1, where 0 indicates an ideally optimal design and 1 indicates a worst design. We take performance measures 1 to 3 as hard constraints. Once any of the constraints is violated, the normalized fitness value will be set to 1. Performance measures 4 and 5 will take part in the objective function definition.

For the design problem above, we formulate the four fitness evaluation components $f_i (i = 1, 2, 3, 4)$ as

$$\begin{cases} f_1 = |A_{\mathrm{ff}} - 0.25| \\ f_2 = |E_{\mathrm{p}}| \\ f_3 = \begin{cases} 1, \text{ if DoF} > 6 \\ \dfrac{\text{DOF} - 1}{5}, \text{ if DoF} \le 6 \end{cases} \\ f_4 = \begin{cases} 1, & \text{if } L_{\min_c} > 0.2 \\ \dfrac{0.2 - L_{\min_c}}{0.2}, & \text{if } L_{\min_c} \le 0.2 \end{cases} \end{cases}$$

where A_{ff} is the flat-folded state profile area, E_{p} is the displacement of the CoM, and $L_{\min_c}$ is the length of the shortest crease. The first two components (f_1 and f_2) evaluate the performance of a solution with respect to the objectives of realizing a flat-folded profile shape area of 0.25 and a minimal displacement of the CoM after being folded. The third and fourth components (f_3 and f_4) provide the measurements for estimating how well a solution satisfies performance measures 4 and 5. In this research, the four fitness components are assigned with a weight vector of $\{0.28, 0.41, 0.1, 0.21\}$ to derive the normalized fitness evaluation F, or say, the problem's objective function:

$$F = 0.28f_1 + 0.41f_2 + 0.1f_3 + 0.21f_4$$

Using the hybrid fitness ranking with the four above components, the GA determines a design with 2 vertices, 9 creases, and 8 faces as in Fig. 5. 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 $L_{\min_c}$ of 0.2119, its fitness vector is $\{1.3353 \times 10^{-4}, 9.6957 \times 10^{-5}, 0.2, 0.0595\}$.

The design shown in Fig. 5 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. 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 Fig. 6, which have very similar performance to the design shown in Fig. 5 on all considered measures?
(2) Before the genetic algorithm actually generates a solution as in Fig. 7 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?

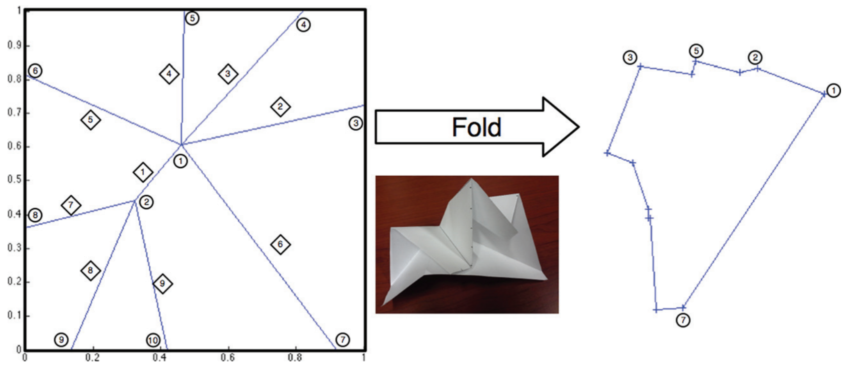

Fig. 5 (*a*) A crease pattern design (normalized fitness of 0.0324) found through the GA for the problem in Sec. 4, where vertices are labeled by numbers in circles, and creases by numbers in diamonds; (*b*) the intermediate folded state; (*c*) the corresponding flat-folded state profile that has an area of about 0.2501. Five vertices are still on the boundary of the profile.

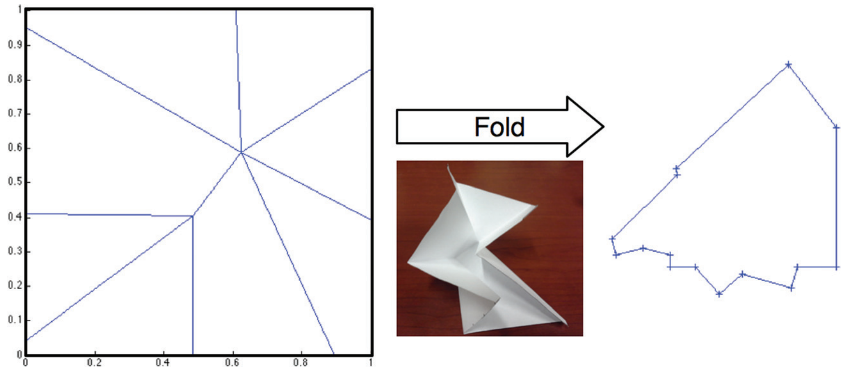

Fig. 6 (*a*) A second crease pattern design (normalized fitness of 0.0323), which is very similar to Fig. 5; (*b*) the intermediate folded status; (*c*) the corresponding flat-folded state profile that has an area of about 0.2499.

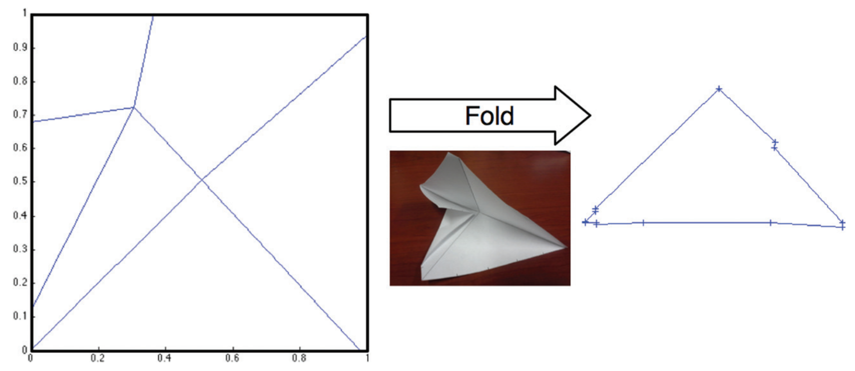

Fig. 7 (*a*) A third crease pattern design (normalized fitness of 0.0035, and fitness vector of {0.0013, 0.0077, 0, 0}), which is almost an ideally optimal design; (*b*) the intermediate folded status; (*c*) the corresponding flat-folded state profile that has an area of about 0.2577

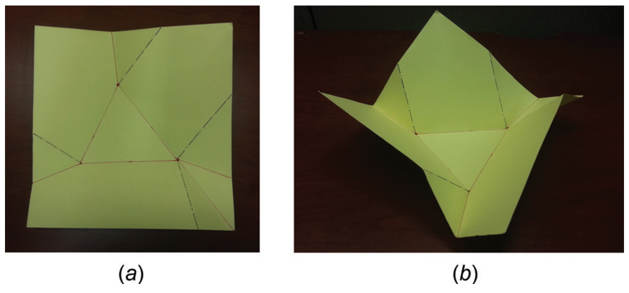

Fig. 8 Fold a square sheet into a bowl. (*a*) A square sheet with a crease pattern, where solid lines are valleys and dashed-dotted lines are mountains. The crease pattern and the crease folding angles are supposed to be designed by GA with ice-cracking. (*b*) The sheet of (*a*) is folded into a bowl/dish shape that can hold water. The shape is required to be rigid-foldable, so that every face keeps flat through the folding.

Nevertheless both the designs in Figs. 5 and 6 have already solved the problem approximately. Of course, the theoretically ultimate target is to get a design like Fig. 7.

5 Conclusion

In this paper, we propose a method called ice-cracking, which is used to generate, encode, and describe origami crease patterns. The ice-cracking method 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 method also serves as a generic geometric representation of origami for computational design methods.

The ice-cracking method 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 method 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 paper, we 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.

The 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, we adapted the GA through balancing the preservation of the elitism and diversity. On the one hand, to protect the elite individuals, we directly lower their probability of getting mutated. Moreover, we 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, we raise the mutation rate for every individual and apply a hybrid fitness ranking to prevent the premature convergence of extreme elitism. The adapted GA with ice-cracking has been implemented and tested on an origami design problem that optimized several geometric and functional properties simultaneously.

The research so far enables the realization and optimization of flat-folded state properties of origami shapes. Our future study will focus on synthesizing the consideration of non-flat folding crease angles into our current algorithm. An example of a problem such as this is to design a crease pattern, which folds a square sheet into a bowl/container (like the shape shown in Fig. 8) capable of holding a desired amount of water. For this problem, the GA with the ice-cracking is supposed to derive a rigid-foldable (foldable without any face being crumpled, bended, or damaged) crease pattern and the dihedral folding angle of each crease at the folded state. The additional concerns related with the problem include how to guarantee or examine the rigid-foldability of candidate solutions, how the ice-cracking ensures the existence of a bottom face for the folded bowl shape to stand on without sacrificing the method's comprehensiveness in representing all possible designs, as well as how the GA consolidates the crease folding angle design into the crease pattern design. Moreover, the inclusion of the folding angle design essentially increases the dimension of the search space of a problem, and thus potentially extends the problem-solving time cost.

References

[1] Harbin, R., 1997, *Secrets of Origami: The Japanese Art of Paper Folding*, 3rd ed., Dover Publications, Mineola, NY.
[2] Gantes, C. J., 2001, *Deployable Structures: Analysis and Design*, WIT Press, Billerica, MA.
[3] Demaine, E. D., Demaine, M. L., and Ku, J., 2011, "Folding Any Orthogonal Maze," Fifth International Meeting of Origami Science, Mathematics, and Education.
[4] Demaine, E. D., and O'Rourke, J., 2007, *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*, Cambridge University Press, Cambridge, UK.
[5] Hull, T., 2006, *Project Origami: Activities for Exploring Mathematics*, A.K. Peters, Natick, MA.

[6] Lang, R. J., 1996, "A Computational Algorithm for Origami Design," 12th Annual ACM Symposium on Computational Geometry, Philadelphia, PA, p. 8.
[7] Lang, R. J., 2003, *Origami Design Secrets: Mathematical Methods for an Ancient Art*, A.K. Peters, Natick, MA.
[8] Tachi, T., 2006, "3D Origami Design Based on Tucking Molecule," Fourth International Meeting of Origami Science, Mathematics, and Education, Pasadena, CA, pp. 259–272.
[9] Tachi, T., 2010, "Origamizing Polyhedral Surfaces," IEEE Trans. Visualization Comput. Graphics, **16**, pp. 298–311.
[10] Mitchell, M., 1996, *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA.
[11] Gallagher, K., and Sambridge, M., 1994, "Genetic Algorithms: A Powerful Tool for Large-Scale Nonlinear Optimization Problems," Comput. Geosci., **20**, pp. 1229–1236.
[12] Mitani, J., 2011, "A Method for Designing Crease Patterns for Flat-Foldable Origami With Numerical Optimization," J. Geom. Graphics, **15**, pp. 195–201.
[13] O'Rourke, J., 2011, *How to Fold It: The Mathematics of Linkages, Origami, and Polyhedra*, Cambridge University Press, Cambridge, UK.
[14] 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, Aug. 4–7, ASME Paper No. DETC2013-13231, p. V06BT07A041.
[15] Poma, F., 2009, *On The Flat-Foldability Of A Crease Pattern*. Available online at: http://poisson.phc.unipi.it/~poma/Ffcp.pdf.
[16] Schneider, J., 2004, *Flat-Foldability of Origami Crease Patterns*, Available online at: http://www.organicorigami.com/thrackle/class/hon394/papers/SchneiderFlatFoldability.pdf.
[17] MathWorks, Digital Modulation.
[18] Jensenm, M. T., 2003, "Reducing the Run-Time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms," IEEE Trans. Evol. Comput.,**7**, pp. 503–515.
[19] Zitzler, E., and Thiele, L., 1999, "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach," IEEE Trans. Evol. Comput., **3**, pp. 257–271.
[20] 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, pp. 98–105.
[21] Konak, A., Coit, D. W., and Smith, A. E., 2006, "Multi-Objective Optimization Using Genetic Algorithms: A Tutorial," Reliab. Eng. Syst. Saf., **91**, pp. 992–1007.
[22] Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Professional, Upper Saddle River, NJ.