2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

Rigid 2D space-filling folds of unbroken linear chains

Zhong Li, Devin J. Balkcom, *Member*, *IEEE*, and Aaron M. Dollar, *Member*, *IEEE*

Abstract— **This paper presents an algorithm for folding a serial revolute chain into a rigid structure of essentially any desired planar shape. The algorithm is fast (linear in the number of links), and the constructed folding plan only requires an actuation method that sequentially folds triangles as the pattern is laid out, maintaining incremental rigidity of the structure during folding.**

I. Introduction

Consider a planar "snake robot" that is a long serial chain of equal-length links connected by revolute joints. Further, the robot is designed to be foldable; connections can be made between any two link endpoints that come into contact. What rigid shapes can be folded out of this robot without causing self-intersection, while limiting the complexity of the actions required during folding?

This paper proves that essentially any planar smooth-boundaried shape that does not have any parts that are "too narrow" or "too curved" with respect to the link length can be approximately covered by a rigid lattice that is easily foldable from a single chain. We present a fast (linear time in the number of links) algorithm that, given a target shape, generates this rigid lattice together with a folding plan to fold the lattice. Further, the algorithm suggests an *on-line* folding strategy for which is not even necessary to know the shape ahead of time, as long as each link has sensors that can detect the boundary of the shape as folding occurs.

A partially-folded snake robot can be considered in two parts: the part that makes up the structure that has already been folded, and the remaining, unfolded part, which we will call the "tail". We do not explicitly consider the problem of self-intersections in the tail – imagine, for example, that the tail is stored in a spool, and laid out into the folded structure during the folding process.

The main contribution of the work is an insight into a particular flavor of modular robotics. The benefits of reconfigurable modular robots have been discussed at some length in the literature; simple subsystems can combine and recombine to form structures that are most suitable for the task at hand. Perhaps the simplest task is to form a rigid structure of some desired shape. Connecting robots made of separate modules can be challenging, however. Modules must locate each other, may need to climb over each other to assemble the desired structure, and must create power and communication links during connections. In fact, some of these challenges have motivated recent work in folding robots out of origami-like sheets [1] and the study of folding serial chains [2].

The serial design is quite convenient. Chains are themselves easy to construct and build. Power flows through the linear chain already, so only physical connections need to be made during assembly. Further, constraints allow easier connections, requiring fewer degrees of control – different modules do not have to find each other. Folding structures from chains also motivates considering long, skinny links, from which lightweight trusses can be folded.

Zhong Li, and Devin Balkcom are with the Department of Computer Science, Dartmouth College, Hanover, NH 03755 USA. (e-mail: Zhong.Li.GR@dartmouth.edu, devin@cs.dartmouth.edu).

Aaron M. Dollar is with the School of Engineering and Applied Science, Yale University, New Haven, CT 06511 USA. (e-mail: aaron.dollar@yale.edu).

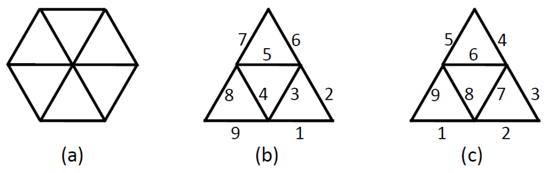

Fig. 1. (a) A hexagon that is not Eulerian, and thus is not foldable from a single chain; (b) A triangle that is Eulerian, with a folding plan that only requires sequential actuation of the joints; (c) A bad folding plan, in which many vertices are nonrigid during folding.

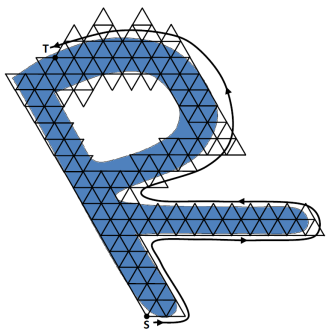

Fig. 2. Our algorithm can approximate a shape of "R" by a rigid triangle lattice folded from a linear single chain and ensures incremental rigidity during folding.

We envision lightweight deployable structures for space and underwater applications, which can be compactly stored in chain form and spooled out as needed. We also envision structures that can be automatically folded to support or repair structures at the macro-scale for engineering and at smaller scales for medical robotics or microbiology. The current paper is still quite far from these eventual applications, but we hope that it lays some groundwork, and identifies some of the interesting geometric problems that must be studied to make progress towards these applications.

Not every geometric configuration of links can be folded from a serial chain. The triangle in Figure 1(b) can be folded using the sequence shown (the first link in the serial chain is labeled 1, and the last is labeled 9), with links intersecting only at endpoints. On the other hand, the hexagon shown in Figure 1(a) cannot be folded from a single chain without placing multiple links at the same location.

Although one can show that 1(a) cannot be folded by exhaustive enumeration of fold sequences, there is also a nice necessary (and sufficient) condition for foldability. Consider the graph with edges along links. There must be a path that visits each link exactly once; the graph must be *Eulerian* or *sub-Eulerian*[3]. It is easy to test if a graph is Eulerian: every vertex must have even degree. In a sub-Eulerian graph, exactly two vertices must have odd degree; these vertices must be the endpoints of the folded serial chain.

Even though it is easy to check if a structure is Eulerian, or to construct an Eulerian structure to approximate any desired shape, there are typically many ways to fold Eulerian structures, and some are better than others. The folding sequence in Figure 1(c) leaves much of the structure non-rigid during folding. Links 1, 2, 3, 4 and 5 must be carefully arranged before finally creating a small triangle when link 6 is folded; we expect this folding strategy to require many "hands" to ensure that the first 5 links stay in place during manipulation.

On the other hand, the folding sequence in Figure 1(b) first constructs the rigid triangle 1-2-3, and then adds the rigid triangle 3-4-5, then 5-6-7, and finally 4-8-9 – a much simpler folding strategy, requiring only a few hands (or otherwise-controlled degrees of freedom).

One could imagine generating an Eulerian lattice of the desired shape and searching the graph for Eulerian paths with good folding characteristics. However, this approach would seem to be too computationally intensive to apply to structures with more than a few dozen links. Therefore, we take an alternate approach, and build structures that are foldable *by design*; elementary substructures that are known to be rigidly foldable (essentially, *fold primitives*) are logically combined to generate larger structures of a desired shape. Because no search is required, the complexity of finding a fold sequence is linear in the number of links in the serial chain.

Figure 2 illustrates a more interesting example than those in Figure 1. Our algorithm first approximates the target shape "R" by a triangle lattice, which is a sub-Eulerian graph. Then the algorithm folds the triangle lattice from a single chain. Basically, this is a problem about how to produce a rigid lattice to cover the target shape and then traverse the lattice in order to fold it. As shown in Figure 2, our algorithm folds the target shape while traversing along the boundary of the shape from the vertex S to the vertex T. During the folding, we maintain a minimum number of nonrigid vertices.

II. Related Work

Modular Robots: The area of modular robotics has a fairly long history; a good starting point for background in this area includes [4], [5], [6], [7], [8], [9], [10], [11], [12]. Closest to our own work is the thesis by Griffith [13], in which it is demonstrated that a linear chain of vertex connected squares can fold into any 2D pixelated shape, and a linear chain of edge connected right-angled tetrahedron can produce an arbitrary 3D voxelated structure. Griffith's approach is to first generate a spanning tree for any given shape, and then convert the spanning tree to an Eulerian path. Based on this result, White and Yim [2] devised a 3D chain modular robot, the *Right Angle Tetrahedron Chain Externally-actuated Testbed*, with the flexibility to form arbitrary space-filling 3D shapes. Our work differs from this previous work in that we explore lightweight serial chains composed of long rods.

Snake Robots: The snake robot is a kind of self-reconfigurable modular robots. Due to the ability to navigate in highly variable environments, snake robots have wide applications. Wright *et al.* [14], [15] designed a modular snake robot, which consists of many fully enclosed actuators.

The Coverage Problem: Perhaps surprisingly, the algorithm we present shares strong similarities with algorithms for mobile robot coverage problems; instead of exploring the space with a robot trajectory, our goal is to cover the space with the robot structure itself. Acar and Choset [16], [17] present a coverage technique based on a hybrid topological and geometrical structure, termed a Morse decomposition. The Morse decomposition approach is complete, because it divides the space into non-overlapping cells and visits each cell by simple motion planning. There are several ways to generate a complete coverage. Acar and Choset [18] devised an algorithm that guarantees the completeness by using critical points. Choset [19] exploits a geometric structure termed exact cellular decomposition; Huang [20] devised an optimal line-sweep-based decomposition achieving the minimum sum of subregion's altitudes. For more results regarding the coverage problem, refer to the review written by Choset [21].

Origami Structures: Rus [1] introduced a method to fabricate a robot from a flat sheet. This idea is inspired by origami, and is similar to our problem. Demaine [22], [23] developed an algorithm for folding a piece of paper into any orthogonal maze with a small scale factor.

Computational Geometry: The manipulation of serial chains has drawn some attention from the computational geometry community. Connelly *et. al* [24], for example, showed that the free configuration space of a serial chain is a single connected component – a carpenter's rule can be folded between any two configurations without self-intersection.

III. Geometric components for building lattices

It is apparent that not all shapes can be equally well-covered by a folded snake robot. Shapes with multiple components, or features that are much smaller than the length of the links of the snake, can clearly not be folded well. We will consider a space of shapes that can guaranteeably be covered with a rigid lattice. We will also define the building blocks that are used to build structures that can be folded incrementally, without too many degrees of control.

First, what shapes can be approximated well by a lattice? In the well-studied computer vision problem of reconstructing a smooth shape from a set of discrete, sampled points, the difficulty is that the problem is under-constrained; very complicated curves could have generated the sampled points. A typical solution is to assume that the shape is relatively simple, without narrow sections or regions of very high curvature. Conveniently, this is exactly the set of restrictions on a shape needed to ensure that a lattice approximates the shape well. We therefore borrow the concept of *Local Feature Size* from Amenta *et al.* [25]. The following two definitions are from [25]:

Definition 1: (from [25]) The *medial axis* of a curve F is the closure of the set of points in the plane which have two or more closest points in F.

Definition 2: (from [25]) The *Local Feature Size*, $LFS(p)$ of a point $p \in F$ is the Euclidean distance from p to the closest point m on the medial axis.

Figure 3 illustrates the medial axis of the target shape. We define a function $r(F)$ that, given a curve F, gives the infimum of the Local Feature Sizes over all points in F. We will show that given a value for r for the boundary of a particular shape, a lattice can be folded that closely approximates the shape.

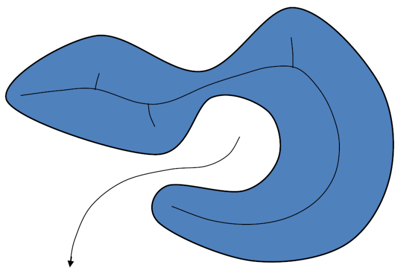
Fig. 3. A target shape with its medial axis.

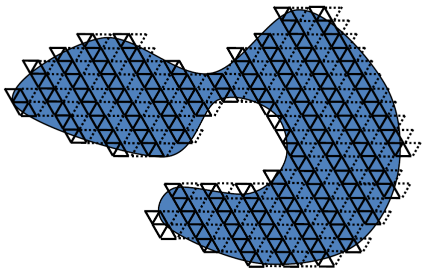
Fig. 5. Generate the k-lattice of the target shape.

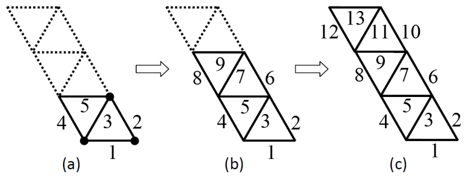

Fig. 4. (a) The folding of one Parallelogram Component, the numbers 1 through 5 indicate the plan of the folding from a single chain; (b) Stack a second parallelogram component to form a strip component of length two by following the pattern shown by the numbers in ascending order; (c) Stack more parallelogram components to form a longer strip component by following the pattern indicated by the numbers in ascending order.

*Definition 3 (***Parallelogram Component***):* The parallelogram component is a parallelogram formed from two equilateral triangles, as shown by solid lines in Figure 4(a).

The parallelogram component is sub-Eulerian and rigid in 2D space. Figure 4(a) illustrates the plan for folding the parallelogram component from a single chain.

In Figure 4(a), we call the three black points the feature points of a parallelogram component, because these three points represent the location of a parallelogram component. When claiming a parallelogram component is inside a shape, we mean that at least one of its three feature points is inside the shape.

*Definition 4 (***Strip Component***):* A strip component is a stack of parallelogram components, as shown in Figure 4(c).

For brevity, we call a strip component a strip hereafter. A strip is a sub-Eulerian graph, and is rigid in 2D space. As shown by Figure 4, we can fold a strip from a single chain incrementally by stacking many parallelogram components.

Now we have a rigid basic structure, a strip, which can be folded incrementally from a single chain. Given a target shape, how can we approximate it by our strips? Our key idea is to first cover the shape by the strips, and then bind them together to form a rigid structure. In order to make sure that we can always bind the strips rigidly, we introduce a rigid structure which we call a triangle lattice as follows:

*Definition 5 (***Triangle Lattice***):* The triangle lattice is a row of strips bound rigidly by *zig-zag* structures (dotted lines in Figure 5).

For brevity, we call each zig-zag structure between a pair of adjacent strips a zig-zag. A zig-zag itself is not rigid, but can bind two strips to form a rigid structure.

Based on the lattice, we can produce a rigid sub-lattice to approximate the target shape. First we put the target shape on the lattice, and then only keep those parallelogram components inside the shape, as shown in Figure 5. The sub-lattice in Figure 5 is not always foldable from a single chain. In section IV, we will show how to add some additional connecting triangles in such a way as to allow folding from a single chain, without worsening the approximate cover by too much.

A. Approximating a shape with a lattice

It is useful to define an approximation measure of how well a sub-lattice approximates a shape.

*Definition 6 (***k-cover***):* Given any finite planar shapes S and T, we say that S *k-covers* T iff no point in T is further than a Euclidean distance of k from S.

We consider a planar lattice to be a shape with points along the edges of the lattice; thus it is meaningful to say that a sub-lattice k-covers some shape and the sub-lattice is termed as a k-lattice of the shape.

We will next show (Lemmas 1 and 2) that the folding produced by our algorithm keeps two significant properties of the target shape:

Geometry: The lattice structure that we fold should have approximately the same geometry as the target shape. The sub-lattice that we fold will cover the shape in such a way that no point in the shape is far from the lattice, and no point in the lattice is far from the shape.

Topology: We want to make sure that holes are not introduced in the lattice that were not present in the shape, and that holes or channels in the shape are not filled in by the lattice. By ensuring that triangles of the lattice are well-connected and fill the original shape sufficiently densely, the folded shape will be sufficiently stiff.

Lemma 1: Given a smooth shape with minimum local feature size $r \in \mathbf{R}^+$, and a lattice with link length l satisfying $l < r/\sqrt{3}$, there exists a sub-lattice cut from the original lattice, such that:

1) The sub-lattice $\sqrt{3}l$-covers the shape.
2) The shape $\sqrt{3}l$-covers the sub-lattice.

Proof: Proof of (1). For each point p on the boundary of the shape, let its nearest point on the medial axis inside the shape be q. Then we have that the distance between p and q is at least $r > \sqrt{3}l$. Let C be the disk with diameter $\sqrt{3}l$ centered on the segment pq and tangent to the shape at p. Then C is entirely inside the shape; otherwise, the nearest point of p on the medial axis is not q, which is a contradiction. Since C has a diameter of $\sqrt{3}l$, it must overlap with some parallelogram component. Then this parallelogram component will be kept in the sub-lattice, and its distance to p is at most $\sqrt{3}l$. Thus each point on the boundary is away from the sub-lattice at most $\sqrt{3}l$. It follows immediately that each point inside the shape is away from the sub-lattice at most $\sqrt{3}l$.

Proof of (2). The shape $\sqrt{3}l$-covers the sub-lattice, because we only keep the parallelogram components with at least one feature point inside the shape. In the worst case, the points

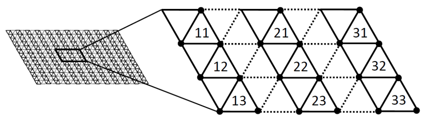

Fig. 6. The coordinate system of the lattice.

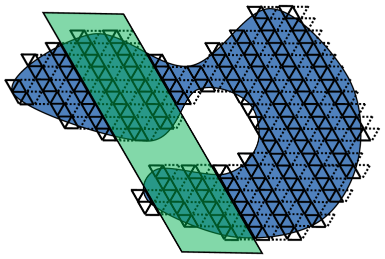

Fig. 7. A slice of the k-lattice.

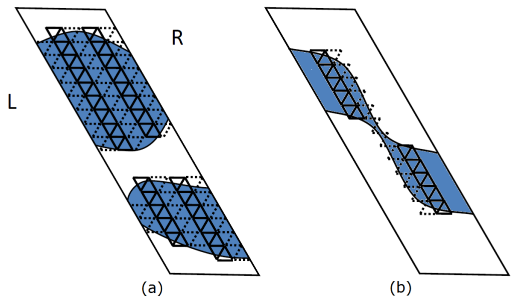

Fig. 8. (a) If the two neighbor strips overlap with each other, they can be bound rigidly by a zig-zag; (b) If the two neighbor strips do not overlap, it implies a skinny part of the target shape.

on the parallelogram can be out of the shape at most the length of the longer diagonal of the parallelogram, which is $\sqrt{3}l$.

■

Lemma 2 will show that choosing the lattice link length as described in Lemma 1 not only allows coverage of the shape, but also ensures that the constructed sub-lattice is rigid.

Lemma 2: Given a target shape with minimum local feature size r, there exists a sub-lattice that l-covers the target shape, where l is the length of the links. This sub-lattice is termed as a l-lattice, which is a bunch of strips. We claim that the l-lattice can be bound rigidly by the zig-zag structures.

Proof: The existence of the l-lattice is proved by Lemma 1. Now we prove that it can be bound rigidly.

As shown in Figure 6, we index each parallelogram component in the lattice by its location in the coordinate system. Thus each parallelogram component is assigned a unique index in the form of (x, y), where x is the row index and y is the column index. In this configuration, a strip is an array of adjacent parallelogram components in the same column.

The sufficient condition for binding two strips rigidly by a zig-zag is that the two strips overlap. The overlapping of two strips should satisfy the following two conditions: (1) they are adjacent with respect to column index; (2) they share common row index. As shown in Figure 7, we highlight a slice of the k-lattice, where two overlapping adjacent strips are bound rigidly by a zig-zag.

It remains to be proven that each strip overlaps with at least one of its neighboring strips. Suppose a strip does not overlap with any of its neighboring strips. This implies that in the target shape, there is a narrow part connecting the isolated strip to other strips, which is illustrated by Figure 8(b). There must be points on the boundary of the narrow part with local feature size smaller than l. But the minimum local feature size over all points on the boundary of the target shape is r and we set the link length $l \leq r/4$. This implies a contradiction. Therefore, every strip in the k-lattice must overlap with at least one of its neighboring strips. It is feasible to bind all strips together rigidly with the zig-zags to form a single piece of the lattice.

■

IV. Folding the Shapes Without Holes

In this section, we will present a fast algorithm for incrementally folding shapes without holes. Both strips and zig-zags are sub-Eulerian graphs; thus they can be folded incrementally. We need to find a plan of folding the strips and zig-zags in order and incrementally. Since the target shape is a closed space in 2D space, if we walk along the boundary counter clock-wise, the target shape is always on our left-hand side. By this fact, we conclude the following key observation for our algorithm.

Observation: For the shape without holes, each strip has both endpoints on the boundary of the shape. Thus by walking along the boundary of the shape once, we can reach every strip twice.

Our key idea of the algorithm follows from the Observation. The algorithm walks along the boundary of the target shape counter clock-wise. Each time it reaches a strip, it folds the strip completely. During the folding, there is at most one nonrigid vertex.

Basically, this walking strategy can be partitioned into three cases: (1) Walk from left to right; (2) Walk from right to left; (3) Climb along the already folded structures.

A. Walk from left to right

Suppose we are folding the strips along the boundary from left to right. We just finished folding one strip, say S_1, and have detected that the next strip to be folded is S_2, which is right-downmost neighbor of S_1. Thus we continue walking from left to right to construct S_2. The way of folding S_2 is shown in Figure 9, by following which we can make sure that there is at most one nonrigid vertex at any time.

There are two cases, shown in Figure 9. If S_2 is not longer than S_1 in the down direction, we follow the folding plan shown in Figure 9(a), in which we start from the bottom of S_2 and go up until completely fold S_2; then we go down through the zig-zag path indicated by the dotted line and back to the boundary. The zig-zag structure is significant. It not only binds S_1 and S_2 together rigidly, but also ensures that we are always on the boundary. In the other case, if S_2 is longer than S_1 in down direction, then we need to follow the folding plan shown in Figure 9(b). In this case, we do not start from the bottom of S_2, but from the parallelogram parallel to the bottom of S_1. Then we folding the upper part of S_2 first. After that, we go down through the zig-zag to fold the lower part of S_2. In this way, we still can fold S_2 completely and back to the boundary.

After folding S_2, we will continue walking along the boundary counter clock-wise. Thus we need to identify which strip is supposed to be folded next. Since we are

Algorithm 1: Main algorithm flow

```
Input: A shape $S$ with $r(S) > 0$
Set the side length $l$ to $r/4$
Generate the $k$-lattice $C$ of the shape (Figure 5)
Start from the left-downmost strip
Initialize the direction indicator $d = 0$
while there exist unfolded parallelograms do
    if d==0 then
        Walk along the boundary Left$\to$ Right
        Let $S$ be the current strip
        if $S$ has not been folded then
            Fold $S$ (Figure 9)
        else
            Climb over $S$
        Search the neighbors of $S$ to be folded
        if $S$ has a left-upmost neighbor $S_L$ then
            Go to the entrance of $S_L$
            Set $d = 1$ (Figure 10(a))
        else if $S$ has a right-downmost neighbor $S_R$ then
            Go to the entrance of $S_R$ (Figure 9)
        else
            Turn around to the left-upmost of $S$ and set d=1 (Figure 10(b))
    else if d==1 then
        Walk along the boundary Right$\to$ Left
        if $S$ has not been folded then
            Fold $S$ (Figure 11)
        else
            Climb over $S$
        Search the neighbors of $S$ to be folded
        if $S$ has a right-downmost neighbor $S_R$ then
            Go to the entrance of $S_R$
            Set $d = 0$ (Figure 12(a))
        else if $S$ has a left-upmost neighbor $S_L$ then
            Go to the entrance of $S_L$ (Figure 11)
        else
            Turn around to the down-right most of $S$ and set d=0 (Figure 12(b))
```

walking from left to right, we follow the order of operations below:

If S_2 has a left neighbor below S_1, then the next strip along the boundary is the left-upmost neighbor of S_2. This claim directly follows the Observation and is illustrated in Figure 10(a).

Else if S_2 has a right neighbor, then the next strip along the boundary is the right-downmost neighbor of S_2. This case follows the Observation and is illustrated in Figure 9.

Else The boundary goes around the right side of S_2 and turn to the left-upmost neighbor of S_2, as shown by Figure 10(b). Then we will walk along the boundary from right to left, which will be immediately introduced in next sub-section.

By following the three steps above in order, we can make sure that we are walking along the boundary and identify the next strip to be folded accurately.

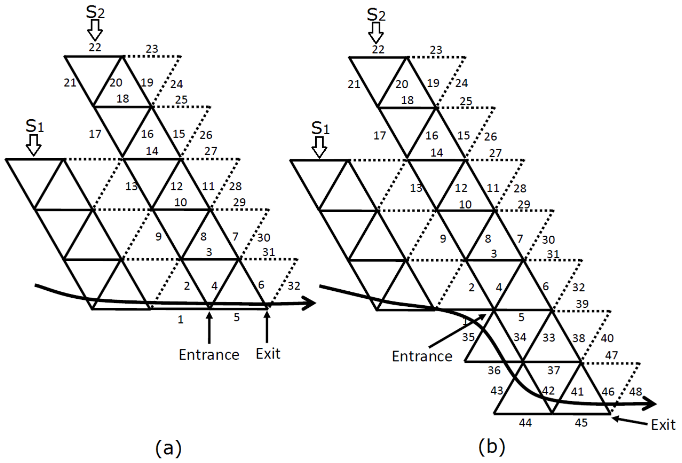

Fig. 9. Walk from left to right along the boundary while folding the strips: (a) If S_2 is not longer than S_1 in the down direction, fold S_2 from the bottom to the top completely; (b) If S_2 is longer than S_1 in the down direction, fold the upper portion of S_2 first, and then the lower portion.

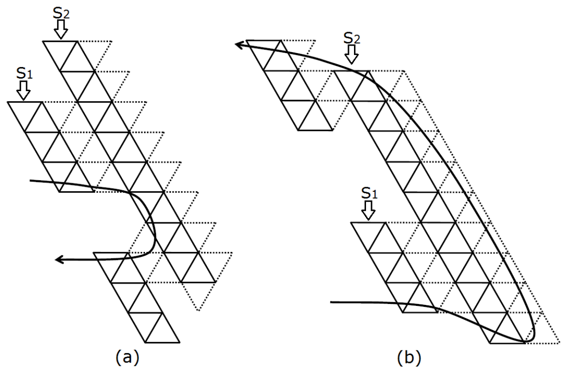

Fig. 10. Detect the boundary while walking from left to right. After folding the strips S_1 and S_2: (a) If there is a upmost unfolded neighbor strip of S_2 below S_1, it is the next strip supposed to be folded along the boundary; (b) If there is no unfolded strip on the left side below S_1 and right side of S_2, it implies that the boundary turn around along the right side of S_2 and then to the left-upmost neighbor strip of S_2.

B. Walk from right to left

Suppose we are folding the strips along the boundary from right to left. We just finished folding one strip, say S_1, and have detected that the next strip to be folded is S_2, which is left-upmost neighbor of S_1. Thus we continue walking from right to left to fold S_2. The way of folding S_2 is shown in Figure 11, in which the folding plan guarantees that there is at most one nonrigid vertex at any time.

There are two cases shown in Figure 11. In both cases, the folding starts from the parallelogram parallel to the top of S_1 and fold the part of S_2 below that parallelogram. In Figure 11(a), the lower part of S_2 is not longer than S_1; we follow the order of the links. In Figure 11(b), the lower part of S_2 is longer than S_1; some extra links are necessary to ensure that at most one nonrigid vertex during the folding. After folding the lower part of S_2, we go up to fold the upper part of S_2, if there is any. We achieve this by climbing along the left side of the already folded structure, as shown by Figure 11. The plan for folding the strips along the boundary from right to left is quite different from that from left to right. But we still can make sure that there is at most one nonrigid vertex during the folding.

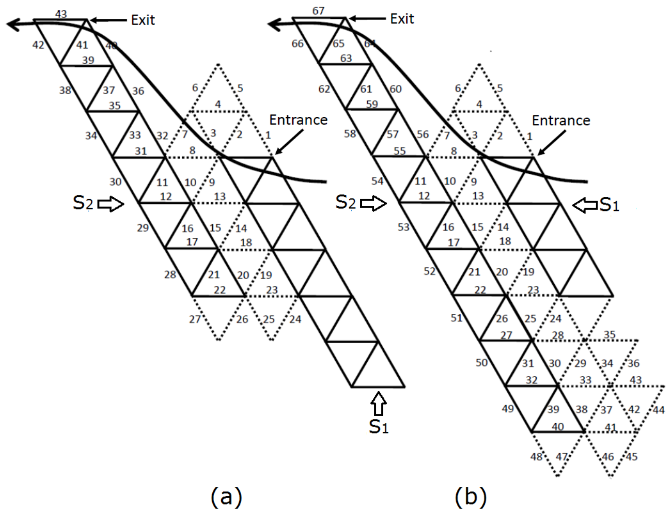

Fig. 11. Walk from right to left along the boundary while folding the strips: (a) If S_2 is not longer than S_1 in the down direction; (b) If S_2 is longer than S_1 in the down direction.

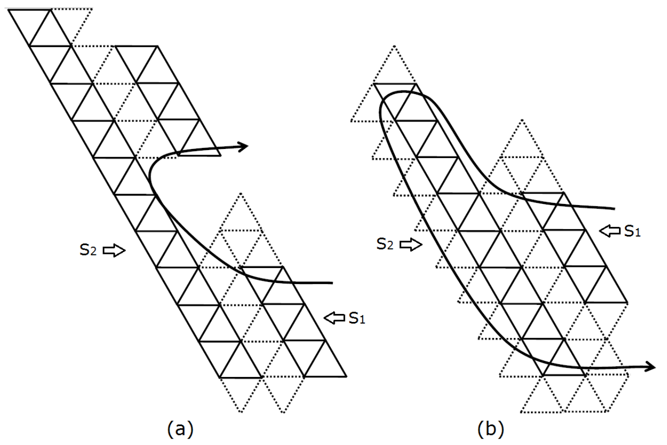

Fig. 12. Detect the boundary while walking from right to left. After folding the strips S_1 and S_2: (a) If there is a downmost unfolded neighbor strip of S_2 above S_1, it is the next strip supposed to be folded along the boundary; (b) If there is no unfolded strip on the right side above S_1 and left side of S_2, it implies that the boundary turn around along the left side of S_2 and then to the right-downmost neighbor strip of S_2.

After folding S_2, we will continue walking along the boundary counter clock-wise. Thus we need to identify the next strip to be folded. Since we are walking from right to left, we follow the order of operations below:

If S_2 has a right neighbor above S_1, then the next strip along the boundary is the right-downmost neighbor of S_2. This claim directly follows the Observation and is illustrated by Figure 12(a).

Else if S_2 has a left neighbor, then the next strip along the boundary is the left-upmost neighbor of S_2. This case follows the Observation and is illustrated by Figure 11.

Else The boundary goes around the left side of S_2 and turn to its right-downmost neighbor. This is shown in Figure 12(b). Then we will walk along the boundary from left to right, which is already introduced in the first sub-section.

By following the three steps above, we can ensure that we walk along the boundary and identify the next strip to be folded accurately.

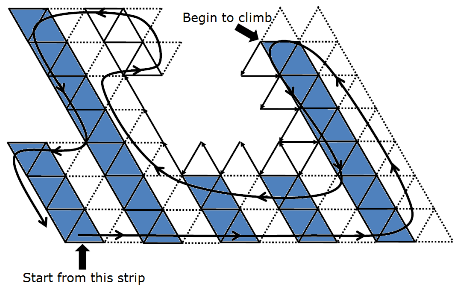

Fig. 13. Climbing over the already-folded structure along the boundary.

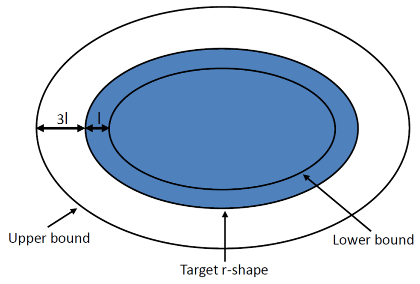

Fig. 14. Approximation bound of the algorithm.

C. Climb along the boundary

Each strip has two endpoints on the boundary. Each time the algorithm reach a strip, the algorithm folds the strip completely. Thus, it is inevitable that the algorithm reaches each strip a second time, and the strip is already folded. In this case, it is necessary to climb over the strip and continue walking along the boundary.

We show this procedure through an example in Figure 13. After generating the k-lattice, we start to construct it from the left-downmost strip, indicated in Figure 13. Then we fold the strips while walking along the boundary counter clock-wise. After folding the rightmost strip, the algorithm turns around and searches along the boundary from right to left. Then, while walking along the boundary from right to left, we find that the strips that we encounter are already folded. Thus we climb over them. This is shown in Figure 13 by a sequence of arrows, which indicate how to climb along the boundary. Finally, we arrive at the last strip that is supposed to be folded. It is always possible to climb along the boundary of the already-folded structure.

V. Approximation Bound of the Algorithm

The algorithm is guaranteed to fold the sub-lattice for the target shape completely. The sub-lattice is at most l smaller than the target shape along the boundary, as shown in Figure 14. While folding the cover, we incorporate some extra structure outside of the sub-lattice in order to ensure that there is at most one nonrigid vertex at any step. Thus the final structure folded by our algorithm is at most $3l$ bigger than the target shape along the boundary, as shown in Figure 14. Therefore, by choosing the length of the links for the lattice to be no larger than one quarter of the minimum local feature of the shape, the lattice structure can be folded without collision; smaller link lengths allow tighter approximation.

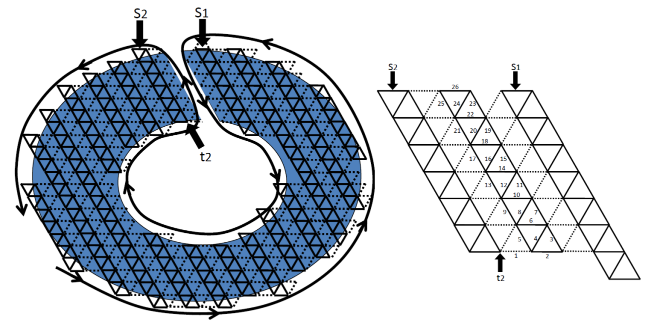

Fig. 15. Converting the sub-lattice of a shape with holes to one without holes. The algorithm folds the lattice while walking along the boundary counter-clockwise, and fills the pipe while climbing out of the hole.

VI. Algorithm for the shape with holes

For the shape with holes, we first generate the sub-lattice. However, it could happen that some strips have no endpoint on the boundary, and therefore could not be folded by our algorithm. To solve this problem, we convert the sub-lattice with holes to one without holes by deleting one strip. We term that strip a *pipe*, as shown by Figure 15. Then we follow the algorithm for folding the shape without holes. We fill the pipe as a final step, while climbing out of the hole.

We illustrate how to fill the *pipe* in Figure 15. By the algorithm, after folding the strip S_1, we will walk into the hole along the dotted triangles on the left of S_1. Then we will fold the strips while walking along the boundary of the hole. After folding strip S_2, the algorithm will climb along the triangles on the right side of S_2 for folding the shape without holes. But now we need to fill the *pipe* by following the folding plan shown in Figure 15.

We just articulated a simple example with one hole. For the sub-lattice of the target shape with many holes, we build a pipe for each of the holes. Then we obtain a sub-lattice without holes. We follow the algorithm for folding the shape without holes. While climbing out of each hole, we block the pipe by following the folding plan illustrated by Figure 15. This achieves our goal of folding the sub-lattice of any shape with holes.

VII. Conclusion

This paper explored a particular method of folding linear chains into rigid planar shapes. The algorithm developed draws some inspiration from algorithms designed for mobile robot area-coverage problems [16], [17]. Although the basic folding strategy is simple (fold sequential strips of triangles), some challenge is introduced by the need to connect strips in such a way as to ensure complete coverage, with a single rigid substructure complete at each stage in the fold, and without folding too far outside of the permitted region.

This paper represents only an initial exploration of an interesting geometric problem, but the problem and approach suggest several very interesting directions of future work. Can 3D trusses be folded on-demand given a target 3D shape? How can folding be accomplished with other methods of actuation; for example, parallel actuation of all joints?

VIII. Acknowledgements

The authors wish to acknowledge Weifu Wang, Corey O'Hern, and Lynne Regan for their insightful comments, input, and feedback. Work on this project was in part supported by NSF grants IIS-0953856 (Dollar) and IIS-1217447 (Balkcom).

References

[1] C. D. Onal, R. J. Wood, and D. Rus, "Towards printable robotics: Origami-inspired planar fabrication of three-dimensional mechanisms," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 4608–4613, 2011.
[2] P. J. White, C. E. Thorne, and M. Yim, "Right angle tetrahedron chain externally-actuated testbed (ratchet): A shape changing system," *Proceedings of International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 7, pp. 807–817, 2009.
[3] F. Jaeger, "A note on sub-eulerian graphs," *Journal of Graph Theory*, vol. 3, p. 91, 1979.
[4] M. Yim, P. J. White, M. Park, and J. Sastra, "Modular self-reconfigurable robots," *Encyclopedia of Complexity and Systems Science*, pp. 5618–5631, 2009.
[5] G. Chirikjian, "Kinematics of a metamorphic robotic system," *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, pp. 449–455, 1994.
[6] P. J. White and M. Yim, "Reliable external actuation for extending reachable robotic modular self-reconfiguration," *International Symposium on Experimental Robotics*, pp. 13–23, 2008.
[7] C. Unsal, H.Kiliccote, and P. Khosla, "I(ces)0-cubes: A modular self-reconfigurable bipartite robotic system," *In Proceedings of SPIE*, vol. 3839, pp. 258–269, 1999.
[8] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji, "A 3d self-reconfigurable structure," *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, pp. 432–439, 1998.
[9] S. Murata, H. Kurokawa, and S. Kokaji, "Self-assembling machine," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 441–448, 1994.
[10] K. Kotay, D. Rus, M. Vona, and C. D. McGray, "The self-reconfiguring robotic molecule," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 424–431, 1998.
[11] D. Rus and M. Vona, "A physical implementation of the self-reconfiguring crystalline robot," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1726–1733, 2000.
[12] R. Fitch, Z. J. Butler, and D. Rus, "The crystal robot: Implementation and demonstration," *AAAI Mobile Robot Competition*, pp. 65–71, 2002.
[13] S. Griffith, *Growing Machines*. PhD thesis, Massachusetts Institute of Technology, 2004.
[14] C. W. III, A. Buchan, B. Brown, J. Geist, M. Schwerin, D. Rollinson, M. Tesch, and H. Choset, "Design and architecture of the unified modular snake robot," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 4347–4354, 2012.
[15] C. W. III, A. M. Johnson, A. Peck, Z. McCord, A. Naaktgeboren, P. Gianfortoni, M. Gonzalez-Rivero, R. L. Hatton, and H. Choset, "Design of a modular snake robot," *Proceedings of the International Conference on Intelligent Robots and Systems*, pp. 2609–2614, 2007.
[16] E. U. Acar, H. Choset, A. A. Rizzi, P. N. Atkar, and D. Hull, "Morse decompositions for coverage tasks," *International Journal of Robotics Research*, vol. 21, no. 4, pp. 331–344, 2002.
[17] E. U. Acar and H. Choset, "Sensor-based coverage of unknown environments," *International Journal of Robotics Research*, vol. 21, no. 4, pp. 345–366, 2002.
[18] E. U. Acar and H. Choset, "Critical point sensing in unknown environments," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 3803–3810, 2000.
[19] H. Choset, "Coverage of known spaces: The boustrophedon cellular decomposition," *Autonomous Robots*, vol. 9, no. 3, pp. 247–253, 2000.
[20] W. H. Huang, "Optimal line-sweep-based decompositions for coverage algorithms," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 27–32, 2001.
[21] H. Choset, "Coverage for robotics - a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1-4, pp. 113–126, 2001.
[22] E. Demaine, M. Demaine, and J. Ku, "Folding any orthogonal maze," *Proceedings of the 5th International Conference on Origami in Science, Mathematics and Education*, vol. 13-17, p. 449, 2010.
[23] E. D. Demaine, M. L. Demaine, and J. S. B. Mitchell, "Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami," *Computational Geometry*, vol. 16, no. 1, pp. 3–21, 2000.
[24] R. Connelly, E. D. Demaine, and G. Rote, "Straighting polygonal arcs and convexifying polygonal cycles," *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pp. 432–442, 2000.
[25] N. Amenta, M. Bern, and D. Eppstein, "The crust and β-skeleton: combinatorial curve reconstruction," *Graphical Models and Image Processing*, vol. 60, no. 2, pp. 125–135, 1998.