Robotica (2011) volume 29, pp. 87–102. © Cambridge University Press 2011
doi:10.1017/S0263574710000731

Planning to fold multiple objects from a single self-folding sheet

Byoungkwon An, Nadia Benbernou, Erik D. Demaine* and Daniela Rus

MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA

(Received in Final Form: November 3, 2010)

SUMMARY

This paper considers planning and control algorithms that enable a programmable sheet to realize different shapes by autonomous folding. Prior work on self-reconfiguring machines has considered modular systems in which independent units coordinate with their neighbors to realize a desired shape. A key limitation in these prior systems is the typically many operations to make and break connections with neighbors, which lead to brittle performance. We seek to mitigate these difficulties through the unique concept of self-folding origami with a universal fixed set of hinges. This approach exploits a single sheet composed of interconnected triangular sections. The sheet is able to fold into a set of predetermined shapes using embedded actuation.

We describe the planning algorithms underlying these self-folding sheets, forming a new family of reconfigurable robots that fold themselves into origami by actuating edges to fold by desired angles at desired times. Given a flat sheet, the set of hinges, and a desired folded state for the sheet, the algorithms (1) plan a continuous folding motion into the desired state, (2) discretize this motion into a practicable sequence of phases, (3) overlay these patterns and factor the steps into a minimum set of groups, and (4) automatically plan the location of actuators and threads on the sheet for implementing the shape-formation control.

KEYWORDS: Modular robots; Multi-robot systems; Motion planning; Control of robotic systems; Design.

1. Introduction

Over the past two decades we have seen great progress toward creating self-assembling, self-reconfiguring, self-replicating, and self-organizing machines.[36] Prior work in self-reconfiguring robots has created modular robot systems with unit size on the order of several centimeters and above,[1,5,9,17–20,22,27,32,37] and supporting algorithms for achieving self-reconfiguration planning and control.[6,10,23,26,30,35] Each of these prior works considers unit-modular systems, in some cases using all identical modules, and other systems using multiple (e.g., two) types of modules.

This paper studies a new type of self-reconfiguring system without modules called a *self-folding sheet*. Self-folding sheets form a new approach to reconfigurable robotics and programmable matter invented by an MIT–Harvard collaboration.[21] A self-folding sheet is a flat piece of material with integrated sensing, actuation, communication, computation, and connectors capable of autonomously folding into desired shapes. It is similar to a piece of paper that can be folded by humans into origami structures, except that the sheet *folds itself* into shape. The self-folding sheet is assumed to be constructed with a built-in structure of bends that can be actuated individually, or with parallel group actuation, to achieve a desired deformation of the sheet that is consistent with the physical constraints of its folding structure. Controlled by either on-board or external computation, the sheet acts on itself internally to transform at will into a desired shape. Each sheet can achieve multiple shapes. For example, the sheet could fold itself into a table, and once the table is no longer needed, the sheet could become flat for re-use at a later time, for example as a chair or a bookshelf, according to the user's needs.

The sheet approach has two advantages over unit-modular self-reconfiguring robot systems. Unlike the usual reconfigurable modular robots, a self-folding sheet is a single morphable robot, simplifying robot construction and assembly by avoiding the need for complex connectors and more than one power source. Furthermore, unlike most robots in general, a self-folding sheet is a thin two-dimensional object, making it relatively easy to manufacture using standard tools for 2D rapid prototyping. The fabrication processes and manufacturing details are described in ref. [21].

In this paper, we present an algorithmic theory for how to design and control self-folding sheets, minimizing parameters such as number of actuators. While a mathematically ideal self-folding sheet can actuate any of the infinitely many possible crease lines by any amount at any time, building such a general sheet would be inefficient if not impossible. In this paper, we develop algorithms to design an efficient construction of a self-folding sheet that can fold into any shape belonging to a desired set of foldings. By specifying the target shapes in this way, we show how to optimize the design to use few actuators and re-use shared components among foldings.

How do we choose foldings that can share many components? A powerful class of origami achieving this goal is *box pleating*, where all creases of a rectangular sheet lie along a square grid with alternating diagonals; see Fig. 1. In mathematics, this grid is called the *tetrakis tiling*. The $n \times n$ box-pleat pattern was recently shown to be *universal*

* Corresponding author. E-mail: edemaine@mit.edu

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

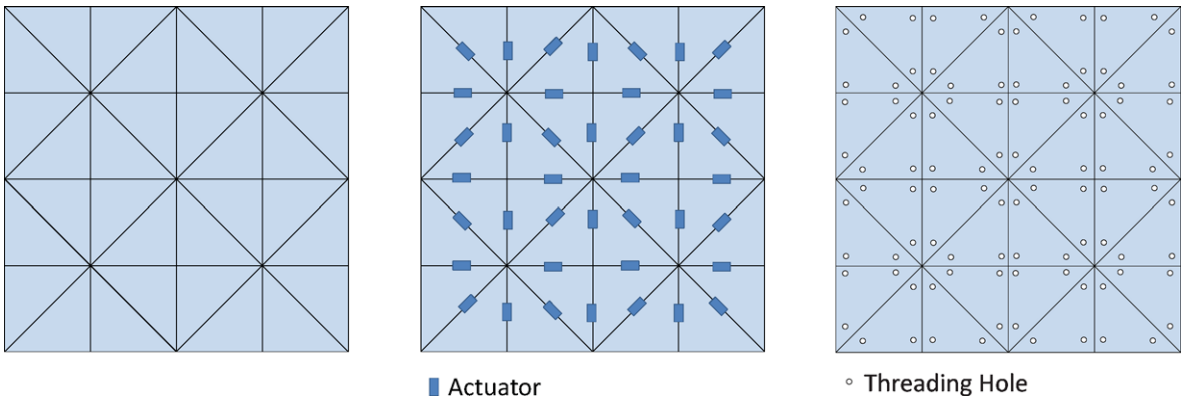


Fig. 1. A 4 × 4 box-pleated sheet (left), with edge actuators (middle) and threading holes (right).

in that crease patterns formed by subsets of the hinges fold into all possible orthogonal shapes made out of $O(n)$ cubes.[4] Therefore, exponentially many shapes can be made from different subsets of one common hinge pattern, forcing this collection of foldings to share many creases. We focus here on box pleating because of its uniformity and versatility for both practical and algorithmic origami design, although our techniques apply more generally to any straightline hinge pattern.

We consider two models of actuators, corresponding to what we have tested in physical self-folding sheets. In the first model (Fig. 1, middle), we place one *edge actuator* along each foldable edge so that the edge may be bent to one of a few possible fold angles. Such an actuator can be implemented by several technologies, such as a flat strip of shape memory alloy (SMA) wire, programmed to bend to a desired angle when heated via electrical current. In the second model (Fig. 1, right), we sew spring SMA wires into the sheet so that several sides may be pulled together at once. This *threading* model can be more economical in terms of the number of actuators, but limits the number of target shapes. Our algorithms detail how to place either type of actuator.

We present and analyze several algorithms for designing self-folding sheets and their plans for folding into a desired set of shapes; refer to Fig. 2. First, we apply our single origami planner (Section 3) to each origami design individually, producing a discrete plan for folding into each single shape. Second, our multiple origami planner (Section 4) combines these plans into a unified construction of one sheet with one plan for each target shape. Third, our threading design planner (Section 5) converts these plans, which apply directly to flat SMA actuators, to work with threaded SMA wire,

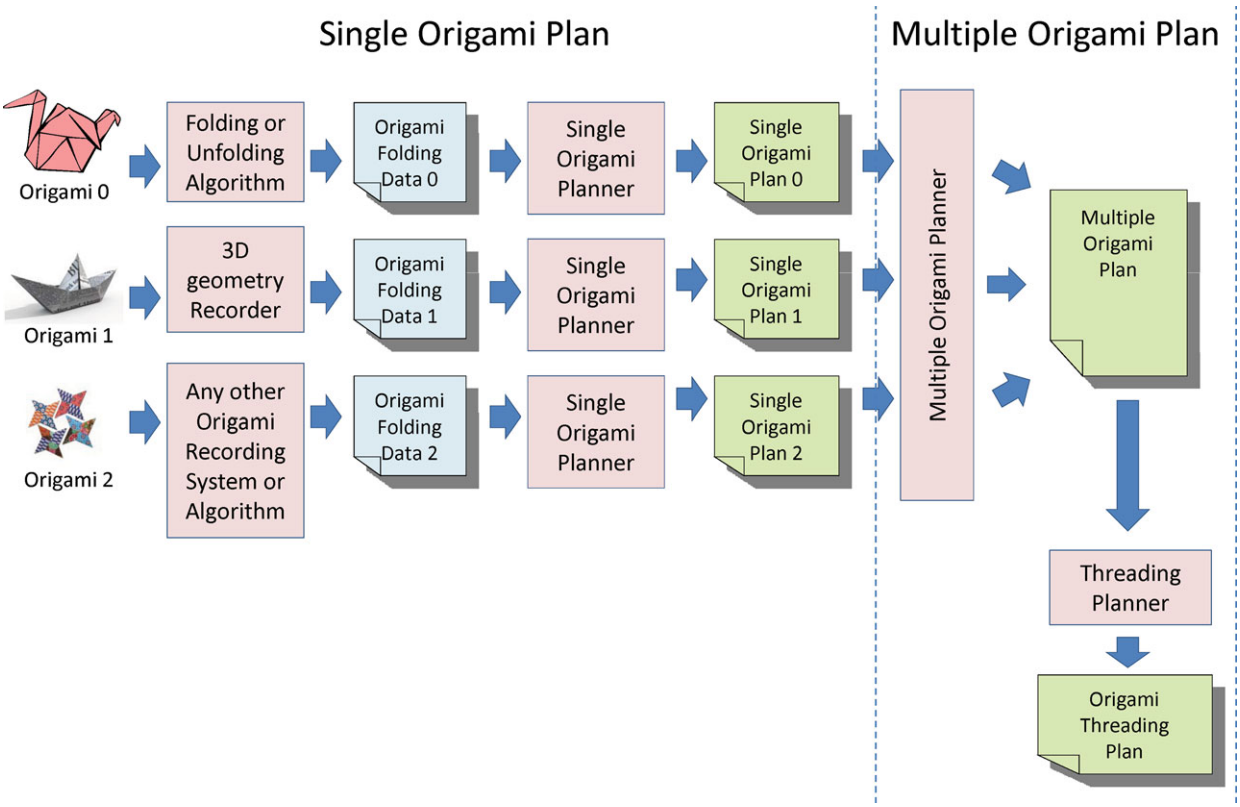


Fig. 2. Visual overview of planning algorithms.

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

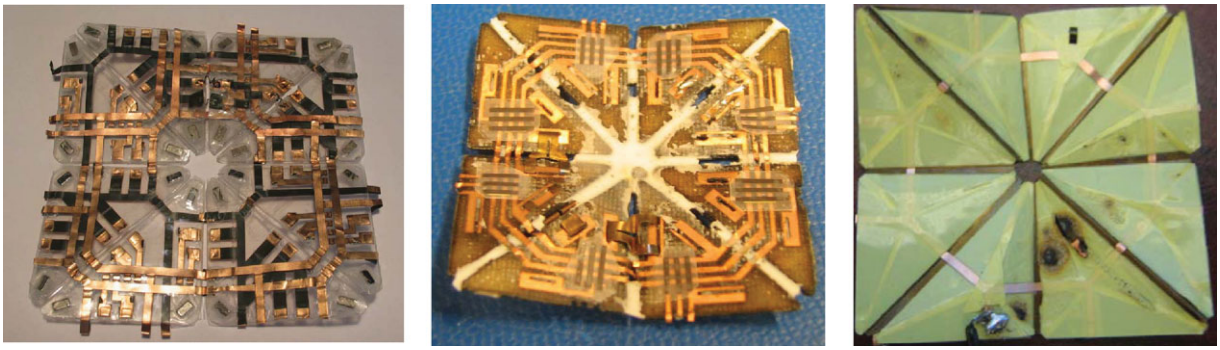


Fig. 3. Three small examples of manufactured self-folding sheets.

often substantially reducing the number of actuators required. Driving this planning process is an outer optimization loop (Section 6) which reduces the number of actuators required by the planners via optimal alignment of the target origami designs.

The algorithms in this paper are currently implemented in simulation. They assume the existence of a self-folding sheet with a built-in crease structure, embedded actuators, and connectors. It is expected that the algorithms run off-board. Hardware designs and control algorithms that can be executed on a physical device are described in ref. [21]. After adding embedded computation and communication on-board the self-folding sheet, we will enable the on-board execution of the fold-planning algorithms in a centralized or even decentralized way.

Related work: Prior work on robotic origami folding considered the design of a robot and supporting algorithms for manipulating a sheet of material into a folded structure. Balkcom and Mason[3,7,8] have constructed a robot that can make a sequence of *simple folds* – folds along a single line at a time – which enables folding a restrictive class of origami models. By contrast, our folds are generally more complicated, involving several simultaneous creases. Many others have considered robots for automatic folding of cartons and packaging; see, e.g., refs. [13, 25]. All of these robots manipulate the object to fold it, relying on actuation external to the object itself. By contrast, in our work, the actuation is internal: the sheet itself is a self-folding robot.

Other work considers folding origami from identical independent units. Nagpal[28,29] developed algorithms and simulations for folding origami from a sheet of identically programmed autonomous units called "cells." She presents a language for instructing the cells to assemble themselves into a global shape. From this language, a variety of shapes are able to be folded using only local interactions among the cells. By contrast, this paper achieves reconfiguration from a single connected robot, which simplifies construction.

2. Problem Formulation

In this section, we describe the model for designing a self-folding sheet robot capable of folding to any shape among a target set of shapes. We also present relevant definitions from origami mathematics, as well as terms specific to our algorithms. Along the way, we mention physical limitations of designing a self-folding sheet robot.

2.1. Model

A self-folding sheet consists of triangular tiles connected together by flexible hinges arranged in an $x \times y$ box-pleated pattern, as in Fig. 1. The tiles are made from rigid material, while the hinges are flexible. Hinges can be folded by actuators, which occupy some space either within a triangle or along a hinge. Figure 3 shows some small (2×2) examples of self-folding sheets that we have manufactured. The current state-of-the-art in physical robot construction, described in ref. [21], is a 4×4 box-pleated self-folding sheet, though we plan to scale up this resolution in the near future.

Edge actuation model: In the *edge actuation model* (Fig. 1, middle), each actuator lies along a single hinge, and can fold that hinge on demand to a specified angle among a small set of possibilities. Specifically, because we are interested in folding orthogonal shapes, we assume that an actuator can fold an edge by an angle of 0, $+90^\circ$, -90°, $+180^\circ$, or -180°. Such an actuator can be implemented by flat SMA, as we have in our experiments, or with other technologies such as hydro pump or piezoelectric actuator.

Each edge actuator has (up to) five electrical inputs, one per possible fold angle. Applying current to one of these inputs causes the actuator to fold the hinge to the corresponding angle. We allow multiple inputs to be permanently connected together electrically into a *group*, so that applying current on one wire (and closing the circuit) causes all connected inputs to trigger simultaneously. Alternatively, we allow the user to trigger multiple inputs simultaneously by applying current to multiple wires at once. Naturally, groups are preferable whenever possible, because they simplify activation and reduce the number of wires to which the user needs to apply current.

Thread actuation model: In the *thread actuation model* (Fig. 1, right), each actuator lies interior to a tile and is attached to one end of a *thread*, which passes through small holes in the tiles, until reaching its other end which is tied to one of the holes. The actuation contracts the thread, effectively pulling its two ends together. By making the thread short and taut, this actuation causes hinges to fold by (nearly) $\pm 180^\circ$. In the simplest case, the thread just connects two holes on opposite sides of a hinge, in which case actuation

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

simulates an edge actuator: if the loop is on the top side of the sheet, it executes a valley fold, and if the loop is on the bottom side of the sheet, it executes a mountain fold. More generally, several hinges can be pulled shut by a single thread, by routing the thread through several holes, resulting in possibly substantial savings in number of actuators.

Out of practical concerns, we place several requirements on thread designs. We require that each thread is short, and thus stays near a single vertex. Thus, we place the holes at corners of tiles, with three holes per triangular tile. We also allow up to three actuators per tile, one next to each edge, to support large actuators or equivalently small tiles. These limitations, as well as the restriction to fold angles of $\pm 180^\circ$, prevent the thread actuation model from being universal. When a threaded design is successful, though, it can be preferable for its efficiency: as defined, threading always has at most as many actuators as edges, and typically it has fewer. Indeed, the first two self-folding sheets produced by our group (one making a tray, and another making a table) used threaded designs, as they were easier to build. For this reason, we develop algorithms to determine whether a shape or collection of shapes can be produced in this model.

Plans: Given k target shapes, we would like to design a self-folding sheet with embedded actuation (of one of the two types) that is capable of folding itself into each of the k shapes. More precisely, the input consists of k origami designs – valid foldings of the box-pleated sheet – that come from either human origami designers or automated design algorithms. The output from our planning algorithms consists of both a physical design and plans for controlling the actuators for the sheet to reach the target origami designs. The physical design consists of the locations and interconnections of the actuators required to realize the design target shapes. Each plan consists of a sequence of discrete *phases*, where several edges fold simultaneously during each phase. We have found this type of plan to be both flexible – most origami cannot be achieved without folding many creases simultaneously – and practical – often many creases can be actuated together (reducing the number of phases and the intrinsically high number of degrees of freedom). Plans assume that the sheet starts unfolded.

2.2. *Definitions from computational origami*

In this section, we introduce some mathematical terminology and background related to origami, the Japanese art of paper folding. See ref. [16] for further background.

A *hinge* (or *edge*) is a line segment drawn on a sheet of material. A *hinge pattern* is a collection of noncrossing hinges, or equivalently, a planar straight line graph. The box-pleat pattern, consisting of a square grid with alternating diagonals, is the hinge pattern considered in this paper. The hinges divide the piece of paper into regions or tiles called *faces*; in the case of the box-pleat pattern, these are right isosceles triangles.

We allow the sheet to locally rotate (fold) around its hinge. The *fold angle* is the supplement of the dihedral angle between the two faces meeting at the hinge, as shown in Fig. 4. If the fold angle is nonzero, we call the hinge a *crease*. The sign of the fold angle determines the crease as

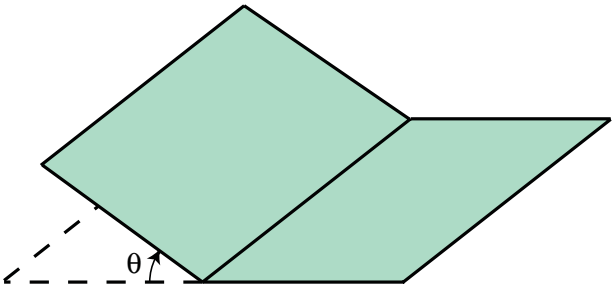


Fig. 4. The *fold angle* at a crease is the supplement of the dihedral angle.

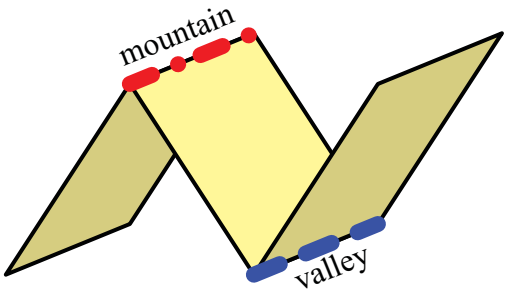


Fig. 5. A crease can be folded as either a mountain fold (left) or a valley fold (right).

either a *mountain fold* or a *valley fold*, as depicted in Fig. 5. We use red lines to indicate mountain folds, and blue lines to indicate valley folds.

A *crease pattern* is a subgraph of the hinge pattern indicating which hinges are actually creases. Figure 6 shows the crease pattern for folding a tray. A *mountain-valley assignment* is an assignment of fold directions (mountain or valley) to the edges of a crease pattern. An *angle assignment* is an assignment of individual fold angles to the edges of a crease pattern. An *origami design* is a crease pattern together with an angle assignment and hence mountain-valley assignment. Such an origami design determines the 3D geometry of the folded piece of paper, called a *folded state*.

A *folding motion* is a continuous folding of a piece of paper from an initial configuration to a target configuration. Typically we start from the flat unfolded configuration, and aim to match the creases and folding angles of a given crease pattern and angle assignment. During the motion, faces are typically allowed to bend when folding origami. In *rigid origami*, the piece of paper is allowed to bend only along hinges; equivalently, the faces can be viewed as rigid panels.

In this paper, we restrict ourselves to origami designs that can be folded rigidly from (a subset of) the box-pleat hinge pattern. Our reasons are that the box-pleat pattern is relatively simple to build, and that the material we use to build the triangular faces of the pattern has high rigidity.

3. Single Origami Planner

The *single origami planner* converts an origami design into a plan for folding the design via a sequence of discrete phases, where each phase consists of one or more creases

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

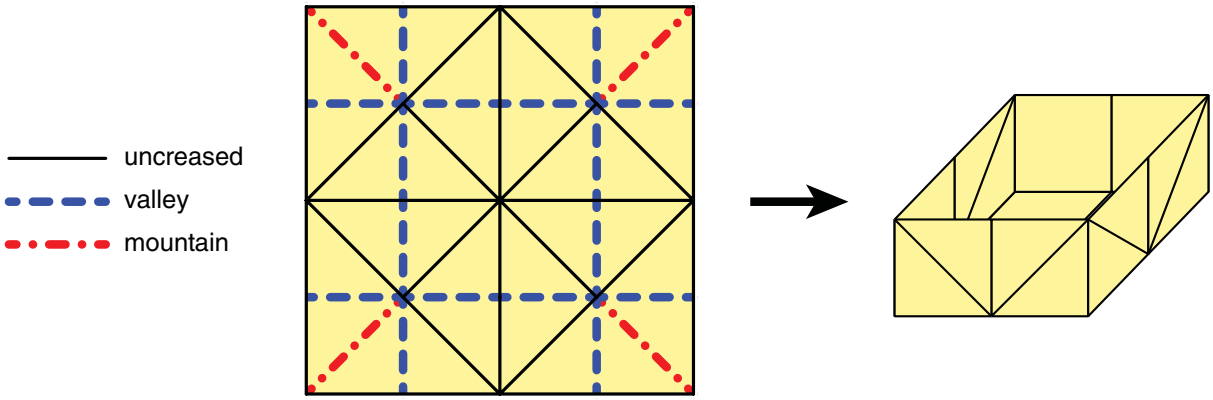


Fig. 6. The crease pattern (left) for folding a tray (right) from a 4 × 4 box-pleated sheet.

folding simultaneously to an angle achievable by an actuator (multiple of 90°). Figure 7 gives an overview of the main steps in the algorithm, each of which we detail below. We assume edge actuation for now, and turn to thread actuation in Section 5.

We have implemented the single origami planner in simulation and generated successful plans for a variety of origami designs – a tray, table, airplane, boat, bench, cup, and elephant – from 4 × 4 and 8 × 8 box-pleated sheets. Figure 8 shows a running example, from the input design in the upper left, to the two phases in the bottom right.

3.1. Folded-state representation

An origami design is most easily specified as a crease pattern together with an angle assignment: which creases get folded by how much. Figure 6 shows a simple example. Thus, this representation is the input for the single origami planner.

There is a unique transformation (up to isometry) of this input into a folded state, that is, the 3D geometry of the folded paper. Step 1 of the single origami planner is to compute this folded state, to obtain an explicit representation of the target 3D shape.

The transformation is standard (often implicit) in computational origami. Perform a depth-first search on the faces of the crease pattern. (Equivalently, this search can be viewed as navigating the vertices of the dual graph.) Place the (arbitrarily chosen) root face r in some canonical fashion, e.g., in the xy plane, with one (arbitrarily chosen) vertex at the origin, and another (arbitrarily chosen) vertex along the x-axis. This placement can be viewed as an isometry (rotation and translation) I_r mapping the face r of the crease pattern into 3D. As the search traverses through an edge e from one face f to another f', we define the isometry $M_{f'}$ of f' as the composition of the isometry M_f of f followed by a rotation around edge e by the (signed) fold angle of the edge. Then we can determine the 3D coordinates of any vertex in the crease pattern by applying a map M_f from a face f containing that vertex; for valid origami designs, it will not matter which such face f we choose.

3.2. Continuous planning

Now that we have a folded state, we need a folding motion from a flat sheet into the folded state. It is known that such a folding motion always exists.[14] This motion, however, may not respect our restriction to rigid origami, where each panel of the hinge pattern remains rigid during the folding. The goal of the continuous part of the single origami planner is to find a continuous folding motion of the desired folded state that respects the rigid origami constraint.

There are many ways to obtain a continuous plan, and the rest of the planning algorithms can use any as input. Perhaps the simplest is to build an instrumented sheet of

Single Origami Planner

1. Given a crease pattern and angle assignment, construct the corresponding folded state (by composing 3D rotations). (This step corresponds to an "instantaneous folding", not a continuous folding motion.)
2. Continuously unfold this folded state using local repulsive energies at each crease (a modification of Rigid Origami Simulator [Tac09]).
3. Reverse the output to obtain a time series of how the crease angles at each edge change as a function of time, starting with a flat sheet and ending with the desired folded state.
4. Discretize each angle time series, approximating it by a step function to minimize the number of steps for a given error tolerance (using a greedy algorithm from [DBM01]).
5. Decompose these steps into discrete phases where several angles fold simultaneously, by splitting at pauses where angles change little.
6. Output a table of target fold angles for each edge during each phase.

Fig. 7. Algorithmic overview of single origami planner.

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

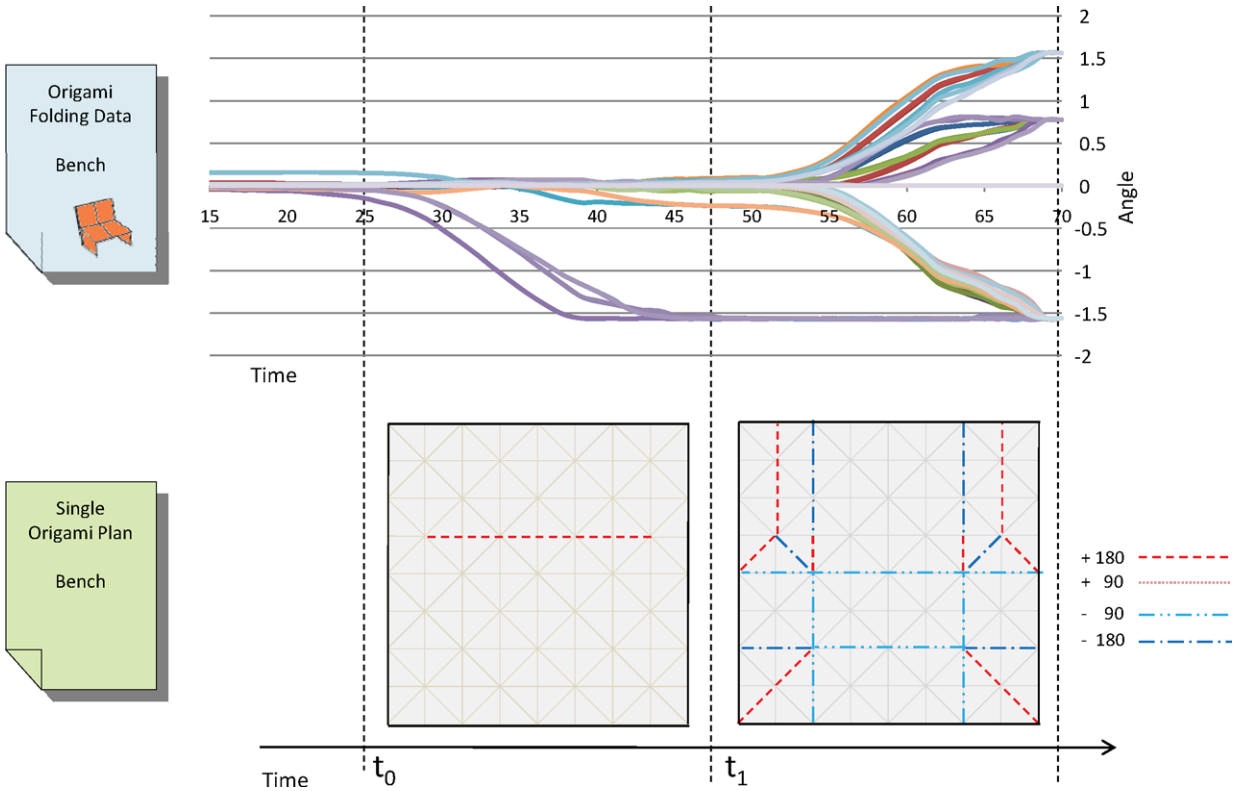


Fig. 8. Example of the single-origami planner applied to a bench design. The top-right plot illustrates the continuous plan, with each curve plotting the fold angle of one of the 176 hinges (divided by 90°) over time. The vertical lines decompose the motion into two discrete phases, in which the creases shown below are active.

paper, and have a human fold it into the desired folded state, with internal or external motion tracking recording the time series of folded states. From this perspective, the continuous motion is simply a more precise form of input to the planner.

Computing a continuous folding motion of rigid origami is relatively understudied, and we believe that additional theory (possibly tailored to the case of the box-pleat hinge pattern) is required to adequately address it. One natural approach is to apply the probabilistic roadmap (PRM) algorithm;[24,31] to our knowledge, this has not yet been attempted for origami crease patterns which have closure constraints on the fold angles of hinges around each vertex. Here, we opt for a (likely faster) heuristic approach, which lacks a guarantee of success, but upon success produces a rigid origami folding motion.

The Rigid Origami Simulator[34] is a software tool designed to simulate the continuous folding of a piece of paper along a specified crease pattern. As the name suggests, the simulation produces folding motions that satisfy the constraints of rigid origami. The Rigid Origami Simulator computes a folding motion by applying local bending energies at the creases and following gradient flow, or more precisely, projecting the unconstrained fold-angle velocities of the edges onto the rigid-origami constraint space specified by ref. [33]. The software, however, does not have an explicit target folded state in mind, and will keep trying to fold all creases until the process converges. It often gets stuck at a configuration different from the intended folded state, and thus not directly applicable to our case.

The author of the Rigid Origami Simulator modified his code for our situation as follows. First, the modified input format allows specifying an angle assignment in addition to a crease pattern. Second, the modified simulator supports *unfolding* from the target folded state, instead of folding from the flat sheet. Neither folding nor unfolding is guaranteed to converge to its goal, but we found unfolding to do so more reliably. Intuitively, we believe that this because the goal state (an unfolded sheet) is most "central" in the configuration space, and therefore easier to reach. An interesting research direction is to investigate and formalize this phenomenon.

When the unfolding algorithm (Step 2 in Fig. 7) converges to an unfolded sheet, we obtain a time series of (partially) folded states, each specified by a different angle assignment to the same creases, which tightly sample a continuous motion from the target folded state to an unfolded sheet. We can then reverse this time series to obtain a motion from an unfolded sheet to the target folded state (Step 3 in Fig. 7). This data is the output of the continuous part of the single origami planner. Figures 9–11 show three examples.

This algorithm is similar to an algorithm for unfolding linkages via local bending energies,[11] for which a pseudopolynomial time bound is known. We suspect that the continuous planner also satisfies a pseudopolynomial time bound, and have found it to be efficient in practice. Table I shows measured running times of an unoptimized implementation.

We have found this continuous planner to succeed on many box-pleated origami models, in fact, on all origami designs we have tested (tray, table, airplane, boat, bench, cup, and elephant). All of these designs are relatively simple (8×8 or smaller), and we suspect that more complex designs will

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

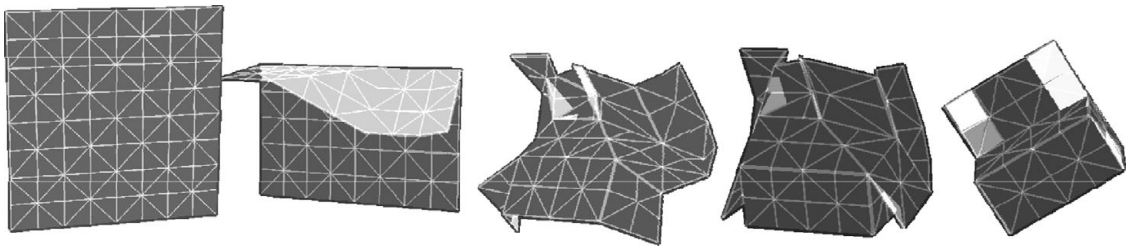


Fig. 9. Continuous plan for a bench computed by unfolding into its 8 × 8 box-pleat pattern.

fail in some cases, based on observing near-failures for some designs. A natural open problem is to characterize when the continuous planner succeeds. Upon failure, the unfolding process will get stuck in a configuration other than the desired flat state, and the user will need to compute or specify a rigid origami folding motion using another method such as those suggested above. The rest of our algorithm works from any continuous folding motion obtained by any of these methods.

3.3. Discrete planning

Our next goal is to discretize the continuous motion computed so far into a short sequence of simply describable steps, so that the motion can be executed by a self-folding sheet robot. This algorithm will run in linear time.

3.3.1. Step function approximation. The first discretization step (Step 4 in Fig. 7) simplifies the motion of each crease separately. The continuous plan tells us the crease's fold angle changes in each time step, which we can view as point samples of a continuous function plotted in angle versus time space. We approximate this function by an optimal step function, where the objective is to minimize the number of steps, and the constraint is that the step function is within a specified additive error ε from the original function at all times.

Díaz-Báñez and Mesa[12] give an optimal $O(n)$ algorithm for this step-function approximation problem, where n is the number of time steps in the input continuous plan. We apply this algorithm repeatedly to obtain the optimal discretization of each crease's continuous motion.

Table I. Approximate running times (averaged over ten runs) for the continuous planner based on Rigid Origami Simulator, including significant overheads in graphical user interface, but excluding load and export times. All designs are represented as 8 × 8 box-pleat patterns.

Airplane	1.20 s		
Boat	1.01 s	CPU:	Intel Core 2 Quad 2.83 GHz (Q9550)
Table	0.97 s	Storage:	3 GB RAM, Seagate 750 GB 300 MBps 7200rpm HD
Bench	8.41 s	Graphics:	NVIDIA Quadro FX 1700
Tray	0.80 s		

For completeness, we briefly describe their algorithm. Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in the plane and an error tolerance ε, plot vertical segments V_i of length 2ε centered at each point p_i. Our constraint that each point be within ε of the step function is equivalent to saying that the step function intersects each of these segments of length 2ε. Sweeping from left to right, the algorithm greedily tries to intersect as many consecutive segments as possible, before starting a new step and repeating this procedure. A vertical segment V_i defines a y interval $[y_i^-, y_i^+]$, where y_i^- and y_i^+ denote the y coordinates of the lower and upper endpoints,

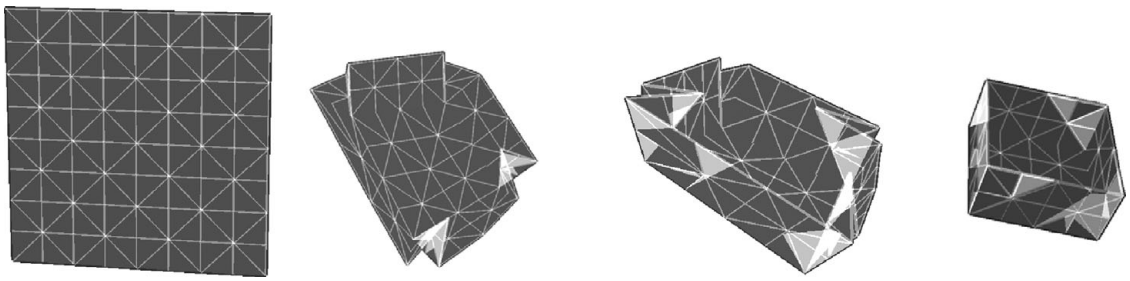


Fig. 10. Continuous plan for a tray computed by unfolding into its 8 × 8 box-pleat pattern.

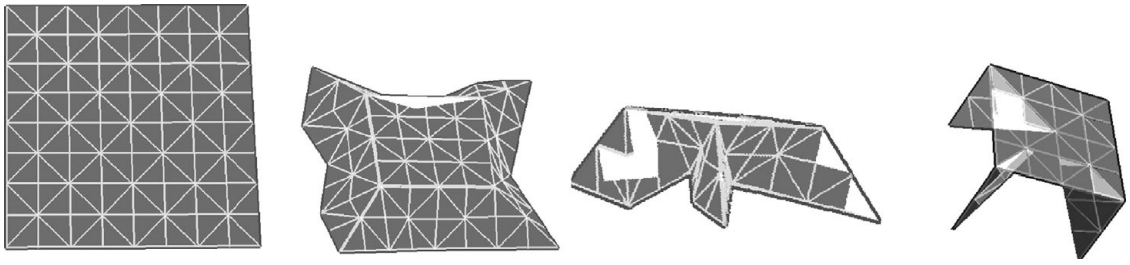


Fig. 11. Continuous plan for a table computed by unfolding into its 8 × 8 box-pleat pattern.

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

Multiple Origami Planner

1. Given n crease patterns, all subsets of a common hinge pattern, and n angle assignments for n origami designs, run the single origami planner on each.
2. Optimize the overlay of these single-origami plans according to a desired objective, via a brute-force algorithm (Section 6).
3. Decompose the phases of the single-origami plans into a minimum set of groups of hinges. (A group is maximal collection of hinges that always move in parallel.)
4. Determine the placement of actuators (and threads if using threading) for each group, to be compatible over all groups.
5. Establish electrical connectivity of actuators within a given group.

Fig. 12. Algorithmic overview of multiple origami planner.

respectively. Sweeping from left to right, we maintain the intersection Δ of the y intervals of the vertical segments until we reach a segment V_i whose y interval does not intersect Δ, in which case we terminate the current step, and start a new step at V_i setting $\Delta = [y_i^-, y_i^+]$. This algorithm runs in $O(n)$ time, and it constructs the optimal step function with error tolerance ε.[12]

3.3.2. Decomposing into phases. At this point, the folding of each edge has been decomposed into a few discrete steps. Our next goal is to coalesce steps from all the edges into *phases*, where a phase moves some subset of the edges together.

We define phases as intervals of time between *pauses*. Intuitively, a pause is a time interval during which all fold angles remain *stationary*. The precise definition of "stationary" requires care, because in the step-function view, an angle is stationary (has zero derivative) at almost every moment in time. We use the heuristic of defining a fold angle to be stationary if its value is either zero or the global minimum or maximum ever taken by the angle during the motion. Given the approximation by step functions and the observation in practice that angles proceed roughly monotonically from zero to the target angle, an angle is effectively stationary if it has not moved much or is near its target value. (This observation is true only for motions produced by the unfolding algorithm, and a different definition of pauses may be necessary for other continuous plans.)

Finally, we assemble the output of the single origami planner. For each phase, we record the final angle of each edge (usually the minimum or maximum) within the phase, rounded to the nearest feasible target angle for an actuator (e.g., multiple of 90°). This table of angles defines the phased folding plan, which serves as input to the multiple origami planner described in the next section.

The running time of this phase-decomposition algorithm, and thus the entire discrete part of the single origami planner, is linear in the total size of the input and output. Table II shows some measured running times of our implementation. The airplane and boat continuous plan inputs have 50 steps, while the table has 30 steps, explaining the faster load for the table.

As this discrete approximation to a folding motion is rather coarse, it needs testing either in simulation or in a real system to ensure that it will succeed in folding the desired folded state. Essentially we are relying on the actuation of the "primary" folds (which fold quickly to a significant angle) to drive the passive "secondary" folds, which all must fold together to realize any nontrivial folding. An interesting direction for future work is to formalize this notion and make guarantees about the discrete resolution required to implement a continuous folding motion.

Table II. Running times (averaged over ten runs) for the single origami planner. All designs are represented as 8×8 box-pleat patterns. Computer specs are the same as Table I.

	Load	Plan	Export	Total
Airplane	571.8 ms	22.1 ms	34.2 ms	628.1 ms
Boat	572.2 ms	23.5 ms	35.7 ms	631.4 ms
Table	404.5 ms	26.6 ms	36.0 ms	467.1 ms

4. Multiple Origami Planner

The *multiple origami planner* combines multiple plans, each produced by the single origami planner, into a single design for a self-folding sheet robot, along with a plan for how to fold the sheet into each of the original origami designs. Figures 13–16 show some two-origami examples, while Figs. 17–18 show some three-origami examples. The multiple origami planner specifies the edge actuators and their electrical connectivity, so it is in fact needed even for designing self-folding sheets that make only a single shape.

The goal of the multiple origami planner is to write each phase in each single-origami plan as a union of groups. As defined in Section 2.1, a *group* consists of one or more edge actuators that can be permanently electrically connected because they always fold together. More precisely, if two folds (including both the hinge and the fold angle) always appear together in each phase of the single origami plans, then they can belong to a common group. Our algorithm essentially maximally packs folds into groups.

The planner requires that the crease patterns for the different origami designs are all subsets of a common hinge pattern, e.g., an $x \times y$ box-pleat pattern. Given origami designs for different resolution box-pleat patterns, such as in Fig. 15 with 8×8 and 4×4 designs, we can simply scale the designs to the least common multiple of the resolutions (8×8 in Fig. 15).

Figure 12 gives an overview of the multiple origami planner algorithm. Step 1 simply runs the single origami planner for each target origami design. Step 2 is an optimization described in Section 6. Step 4 is primarily for

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

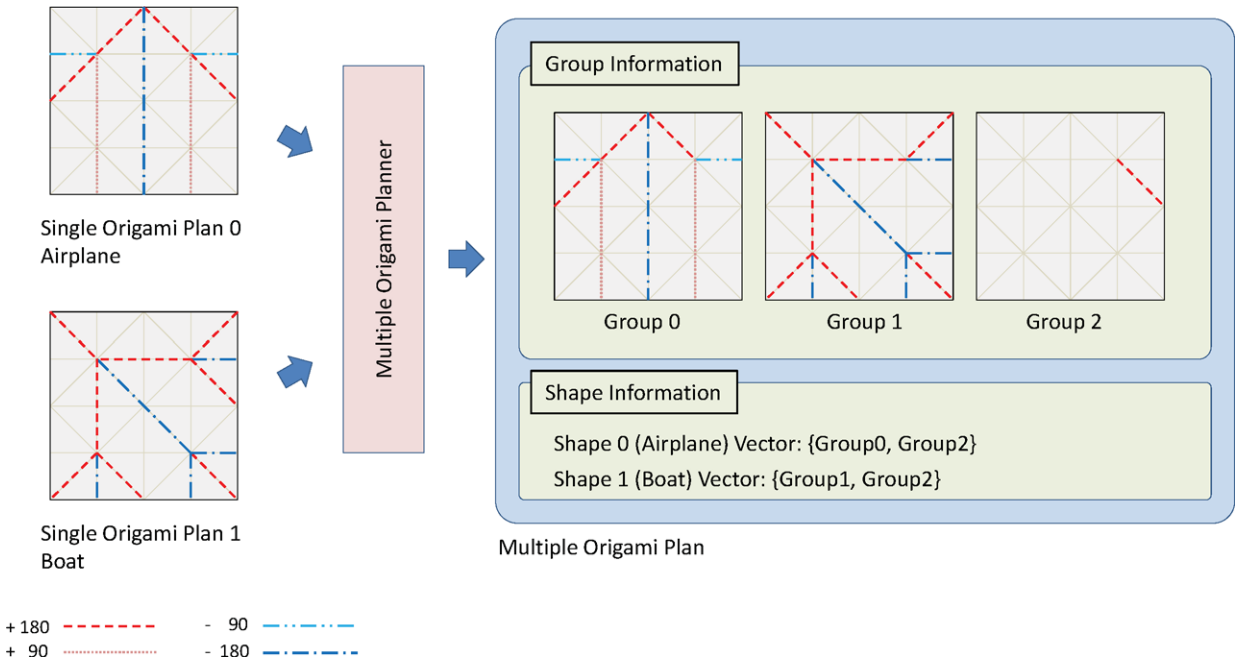


Fig. 13. multiple origami plan for airplane and boat.

threading technology, and will be detailed in Section 5. Step 5 is a standard wire-routing problem, which we do not address here.

The heart of the multiple origami planner is Step 3, which decomposes the phased single-origami plans into groups.

Let $O_1, O_2, \ldots, O_k$ be the given origami designs. We use the term *angled edge* to refer to an edge of a crease pattern with an angle assignment. Let $\hat{E}(O_i)$ denote the set of angled edges of the crease pattern for origami design O_i. (Note that all edges of all crease patterns should be edges of the common hinge pattern; if a crease pattern has what could be interpreted as a longer crease, we split it into its constituent hinges.) Define the *angled union* of the crease patterns of $O_1, O_2, \ldots, O_k$ to be the union of the angled edges over all crease patterns: $\bigcup_{i=1}^{k} \hat{E}(O_i)$.

The multiple origami planner partitions this angled union into the minimum number of disjoint groups such that each phase j of each origami design O_i can be written as a union of groups. (In the worst case, each angled edge can be its own group.) To do this, we define the *signature* of an angled edge e to be the set of pairs (i, j) for which the jth phase of the single-origami plan for O_i activates e. Then we assign one group per distinct signature, consisting of all angled edges with that signature.

For example, in Fig. 13, the only angled crease shared by the two plans is the one that ends up in Group 2. All

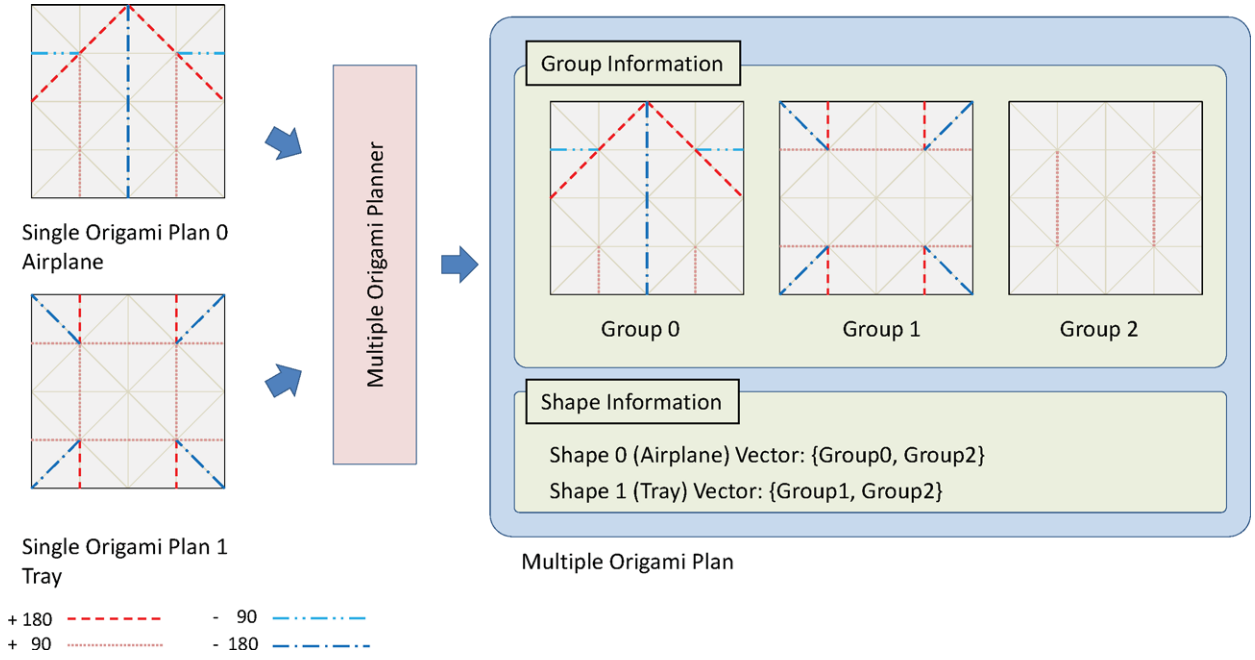


Fig. 14. multiple origami plan for airplane and tray.

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

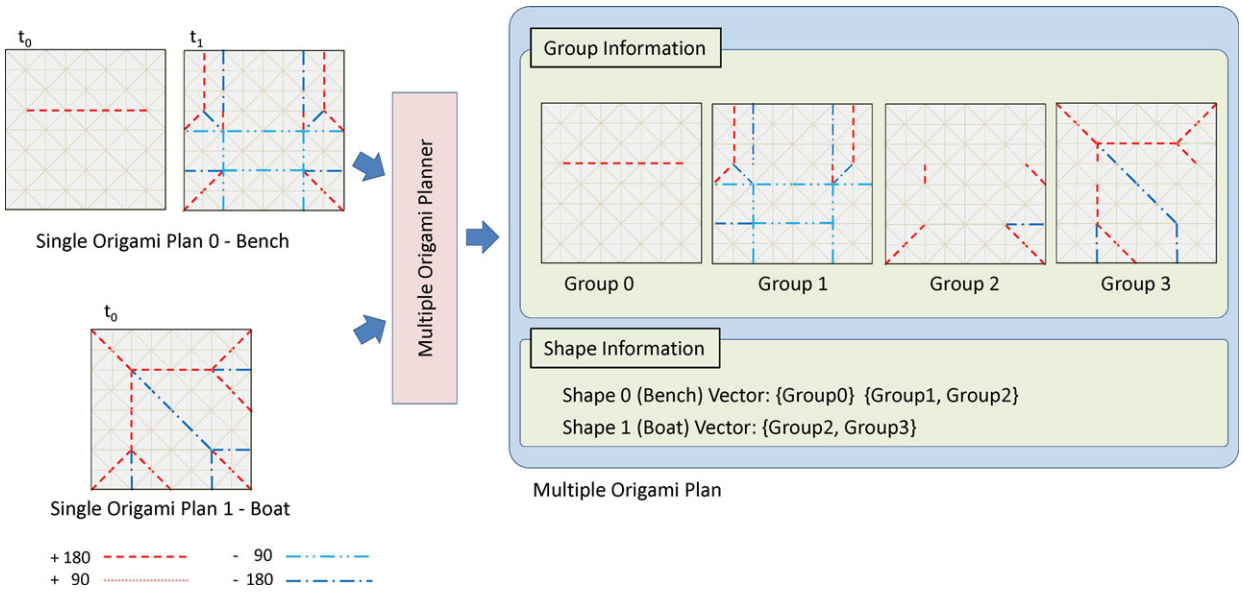


Fig. 15. multiple origami plan for bench and tray.

other creases differ either in location or in fold angle, so they belong to distinct groups. In Fig. 15, some creases appear just in phase 0 of the bench (Group 0), some creases appear in phase 1 of the bench but not in the boat (Group 1), some creases appear both in phase 1 of the bench and in the boat (Group 2), and some creases appear in the boat but not in the bench (Group 3).

To compute this partition into groups, we first loop over every angled edge in every phase j of the single origami plan for O_i, and append (i, j) to the signature of the angled edge. By looping in order, the signatures are already sorted. Now we put the signatures into a hash table to detect duplicate values and thus cluster equal signatures. We can then easily map the matching signatures back to angled edges that belong to a common group. The total running time is linear in the total number n of creases in the input single origami plans.

Theorem 1. *For a set of origami designs $\{O_1, \ldots, O_k\}$, the multiple origami planner produces the minimum possible number of groups that partition the angled union of the crease patterns (for this relative orientation of the crease patterns).*

Proof. If two angled edges e, e' with two signatures s, s' belonged to a common group, then we would only be able to actuate that group during the phases in the intersection $s \cap s'$. But if $s \neq s'$, then at least one phase would not be able to actuate either e or e' as needed, because every angled edge belongs to exactly one group, a contradiction. Thus, only angled edges with equal signatures can belong to a common group, and the algorithm puts all such angled edges together. □

Table III shows some measured running times of our implementation.

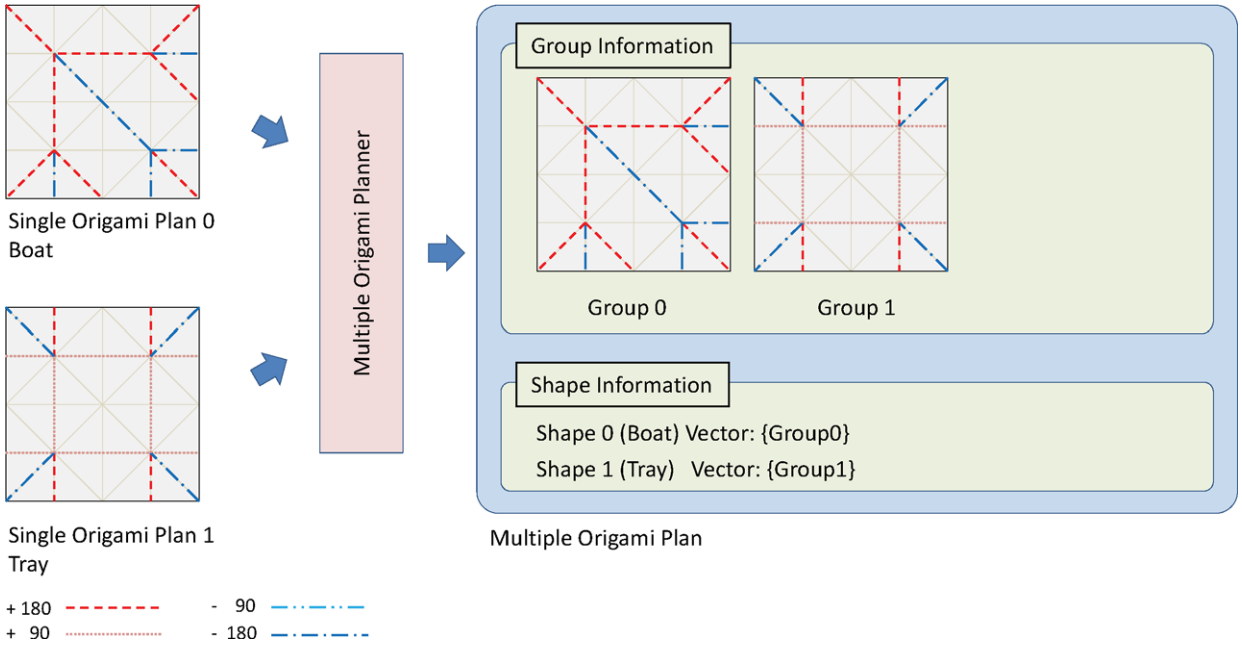


Fig. 16. multiple origami plan for boat and tray.

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

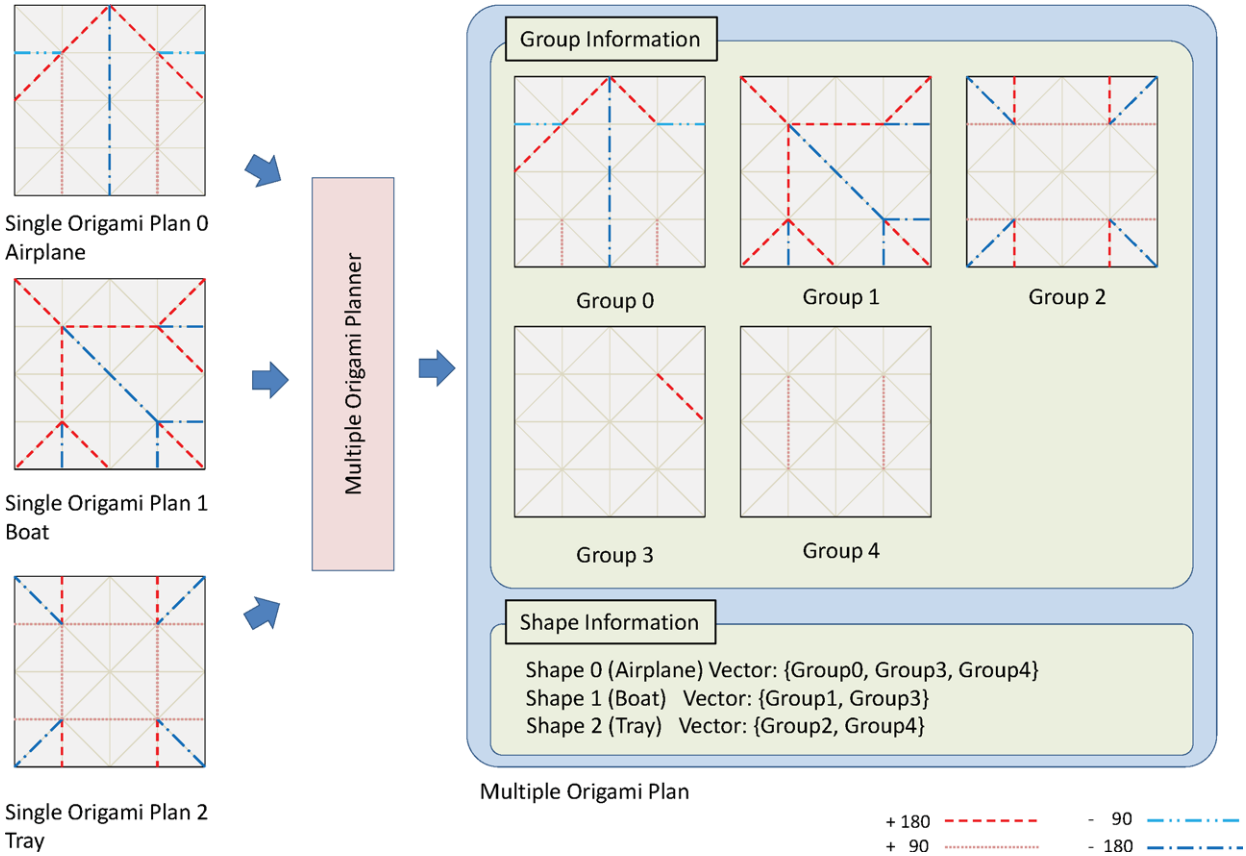


Fig. 17. multiple origami plan for airplane, boat, and tray.

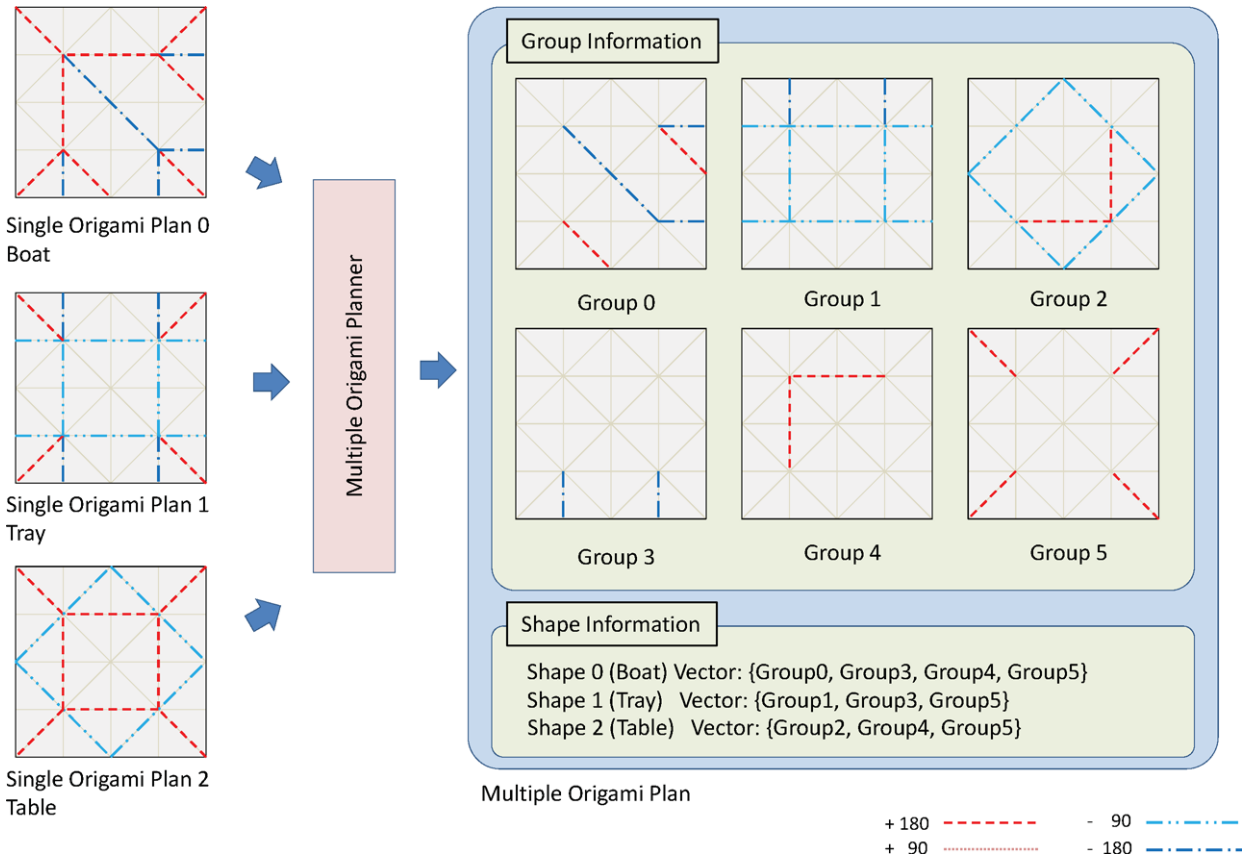


Fig. 18. multiple origami plan for boat, tray, and table.

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

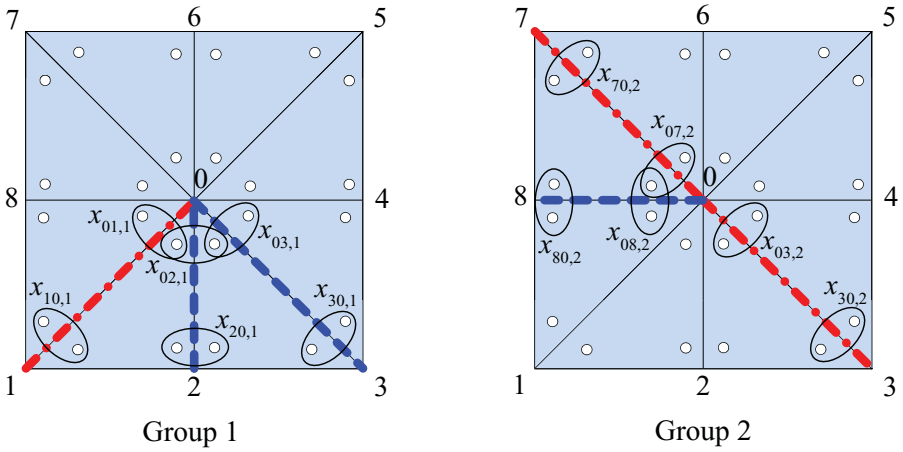


Fig. 19. 2-CNF formula (below) resulting from an example with two groups (above). The top line of constraints are of type (1); the next two constraints are of type (2); and the last three constraints are of type (3).

5. Threading Planner

For the edge actuation model, we simply place an actuator on each hinge that is a crease in at least one of the crease patterns. For the threading actuation model, we use a *threading planner* that converts the result of the multiple origami planner into an actuator placement and thread design. In linear time, the planner either produces such a sheet or determines that none exists.

Because thread actuators can only execute $\pm 180^\circ$ fold angles, the threading planner discards any folds with angles of $\pm 90^\circ$. The resulting simplified multiple origami plan needs to be tested, either in simulation or in a real system, to test whether the more limited actuation suffices to reach the target folded state.

5.1. Threading

Given the output of the multiple origami planner, we construct a 2-CNF formula that is satisfiable if and only if a valid threading exists. Refer to the example in Figs. 19 and 20. Recall that a Boolean formula is 2-Conjunctive Normal Form (*2-CNF*) if it is a disjunction $C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause C_ℓ is the disjunction $x_i \vee x_j$ of exactly two literals (variables or their negations). Such a formula is *satisfiable* if the variables can be assigned values of true or false such that the formula comes out true.

We represent a crease as a pair (u, v) of vertices. For a crease (u, v) folded in group i, we use the Boolean variable $x_{uv,i}$ to denote whether there is a thread connecting the pair of holes adjacent to the crease closest to vertex u. Similarly, $x_{vu,i}$ denotes whether a thread connects the pair of holes closest to vertex v.

We form constraints as follows. First, for each crease (u, v) folded in group i, we add the clause

$$(x_{uv,i} \vee x_{vu,i}) \tag{1}$$

Table III. Running times (averaged over ten runs) for the multiple origami planner, with and without the optimization of Section 6. All designs are represented as 8×8 box-pleat patterns. Computer specs are the same as Table I.

	Unoptimized				Optimized			
	Load	Plan	Export	Total	Load	Plan	Export	Total
Airplane, boat	37.4 ms	1.6 ms	44.4 ms	83.4 ms	34.6 ms	45.2 ms	75.0 ms	154.0 ms
Airplane, boat, table	45.4 ms	1.6 ms	39.0 ms	86.0 ms	45.4 ms	73.4 ms	71.9 ms	190.7 ms

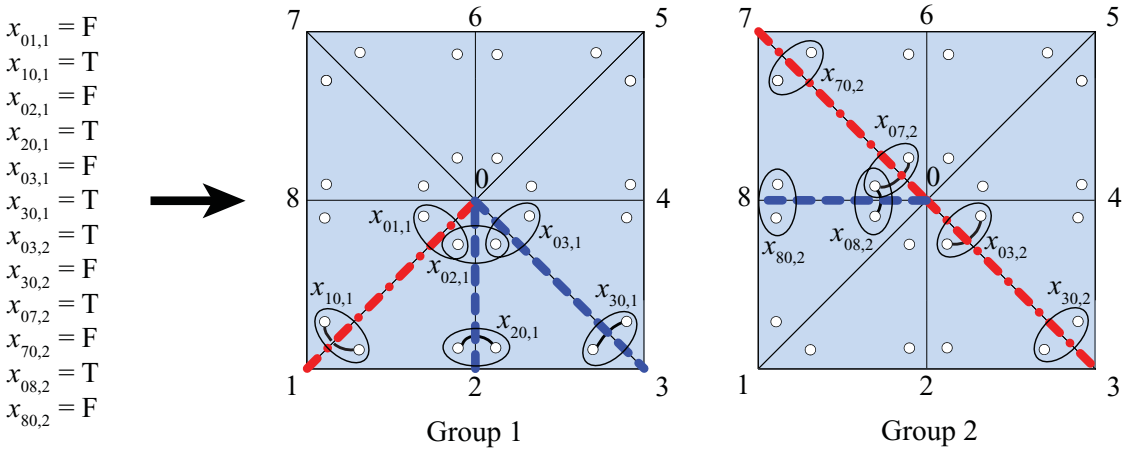


Fig. 20. Converting a satisfying assignment for the 2-CNF formula in Fig. 19 (left) into a threading (right).

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

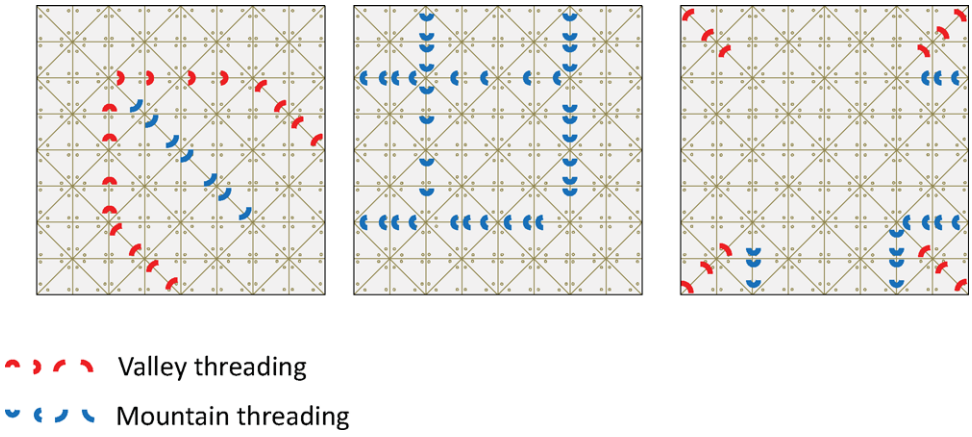


Fig. 21. Output of our threading planner applied to the multiple origami plan for the boat and tray from Fig. 16.

to ensure that the crease gets pulled shut from one side or another, as required by the group. Second, for each crease (u, v) folded in two groups $i \neq j$, we add the clauses

$$(\overline{x_{uv,i}} \vee \overline{x_{uv,j}}) \wedge (\overline{x_{vu,i}} \vee \overline{x_{vu,j}}) \tag{2}$$

to ensure that the same pair of holes is not used in two different groups. Third, for each neighboring pair of edges (u, v) and (u, w) (forming a 45° angle at u), if (u, w) appears in group j, and either $i \neq j$ or (u, v) and (u, w) are both mountain or both valley in group i, then we add the clause

$$(\overline{x_{uv,i}} \vee \overline{x_{uw,i}}). \tag{3}$$

This constraint reflects that two separate threads cannot share a hole, by preventing the use of two hole pairs that share a hole, except in the allowed case where the hole pairs come from creases of opposite direction (mountain and valley) in the same group and thus can belong to the same thread.

Because 2SAT is solvable in linear time,[2] we can find an assignment of variables to satisfy this 2-CNF formula, Φ, or determine that no such assignment exists. Given a satisfying assignment for Φ, we can construct a valid threading as follows; refer to Fig. 20. If $x_{uv,i}$ is set to true, and edge (u, v) is mountain (valley) in group i, then we thread under (over) that edge through the pair of holes closest to u. Also, if we have a consecutive run of edges $x_{uv_1,i}, x_{uv_2,i}, \ldots x_{uv_k,i}$ in group i all set to true, then by constraint (3) these edges must have an alternating mountain/valley assignment in a common group i. Thus, we can concatenate the threadings, weaving alternately over/under each edge.

Theorem 2. *Given box-pleated origami designs $O_1, O_2, \ldots, O_k$ with n total creases, in $O(n)$ time we can either find a threading or determine that none exist.*

Proof. Assume the data structure for a crease (u, v) has pointers to the groups to which it belongs as well as its immediate neighbors. The construction of Φ described above runs in $O(n)$ time, because there are $O(n)$ creases and neighboring pairs of creases, and each takes $O(1)$ time to test and possibly add clauses. In particular, Φ has $O(n)$ clauses. The 2SAT solver[2] takes $O(n)$ time. Finally, given a satisfying assignment, we can construct the corresponding threading as described above in $O(n)$ time. □

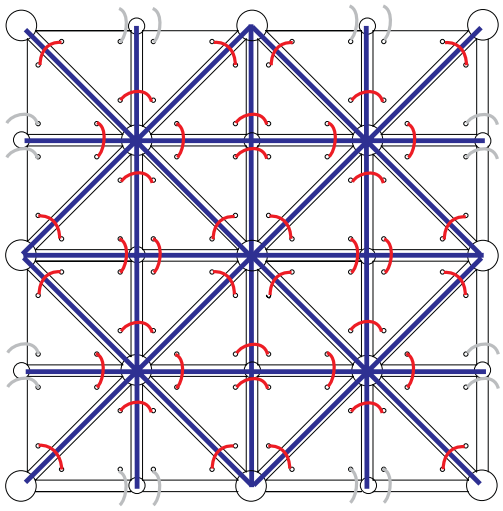


Fig. 22. Threading for the worst-case crease pattern (every crease) on a 4 × 4 box-pleated sheet. Gray threads show how to extend the threading to larger-size sheets.

Figure 21 shows a sample output of our implementation of this algorithm. Although the tray's creases do not all have fold angles of ±180°, we have verified that it folds successfully via threading by building a physical self-folding sheet. This fact naturally leads us to wonder what other shapes with 90° fold angles are possible by threading.

A simple consequence of our characterization of threading is that it is always possible for a single origami design with fold angles of ±180°:

Theorem 3. *Any single box-pleated crease pattern has a valid threading.*

Proof. In the worst-case crease pattern, all hinges are creases and they are all in the same direction, say valley. In this case, we can still construct a valid threading, with one thread per edge, as shown in Fig. 22. Any other crease pattern is a subpattern of this worst case, so we can just use the threads corresponding to edges of the subpattern. □

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

Table IV. Running times (averaged over ten runs) for the threading planner. All designs are represented as 8 × 8 box-pleat patterns. Computer specs are the same as Table I.

	Load + Plan	export	Total
Airplane, boat	42.5 ms	35.5 ms	78.0 ms
Airplane, boat, table	48.0 ms	25.5 ms	73.5 ms

Table IV shows some measured running times of our implementation. The three-shape example likely runs faster because it has no solution to threading.

5.2. *Actuator placement*

Once we have computed a threading of each group from the previous section, it remains to place thread actuators that activate each group. We show that it is enough to consider placing at most one actuator along each edge of a triangle, and with the actuator incident to the threading it activates, because such an actuator placement always exists.

Theorem 4. *Any valid multiple-origami threading has a valid actuator placement in the box-pleat pattern, computable in $O(n)$ time.*

Proof. Each triangular face (u, v, w) of the box-pleat pattern has three holes T_u, T_v and T_w, one in each corner. Any valid threading routes at most one thread from one group in each hole T_i. Consider placing one actuator along each edge of each triangle (the maximum we are allowed). Then we can attach the actuator along (u, v) to the thread through T_u, the actuator along (v, w) to the thread through T_v, and the actuator along (w, u) to the thread through T_w. In this way, every thread has at least one actuator attached to it. We can cull this set of actuators down to just one per thread, at either end of the thread, to obtain a valid actuator placement. To construct this actuator placement in $O(n)$ time, we can simply take each end of each thread, say at the corner u of some triangular face (u, v, w), and attach an actuator along the clockwise-next edge (u, v) of the face. Because this placement is a subset of the full placement described above with one actuator per triangle edge, at most one actuator will be constructed on each edge. □

6. Optimization of the multiple origami planner

In previous sections, we took the origami designs $O_1, O_2, \ldots, O_k$ and their crease patterns to be given in some default orientation. However, the relative orientations of these crease patterns is irrelevant to achieving the desired shapes, leaving us with some freedom in choice. Depending on how we rotate and/or reflect each crease pattern, we can optimize one of several parameters of interest in the final plan:

Number of actuators: Perhaps the most natural objective is to minimize the total number of actuators in the multiple origami plan. For edge actuators, this is the number of edges in the union of the oriented crease patterns; for thread actuators, this is the number of threads produced by the threading planner.

Number of groups: Each distinct group in the multiple origami plan must be connected together by an electrical circuit, with an external input triggering the closure and hence activation of this circuit. Given the limited space for electrical wiring in the sheet, and the desire for few external inputs, we may wish to minimize the total number of groups.

Number of bidirectional edges: It is easier to build an edge actuator that folds in only a single direction (either mountain or valley). Such actuators suffice to make a single origami design, and often suffice to make suitably oriented multiple-origami designs from a single sheet. Such sheets can be found by minimizing the number of edges that need to be folded both mountain and valley in the union of all crease patterns.

We show that each of these optimization problems is *fixed-parameter tractable*[15] in the number k of origami designs. In other words, there is an algorithm with running time $f(k)\,n^{O(1)}$, which is exponential in the parameter k but polynomial in the size n of the input. Specifically, $f(k) \le 8^k$.

Let D_4 denote the dihedral group of the square, i.e., the group of symmetries (rotations and/or reflections) of the square. Each crease pattern has $|D_4| = 8$ possible orientations. Depending on the symmetries of the crease pattern itself (e.g., left-to-right symmetry), there will actually be either one, two, four, or eight distinct orientations of a crease pattern.

Given an orientation $\tau_i \in D_4$ of each origami design O_i, we obtain oriented designs $\tau_1(O_1), \tau_2(O_2), \ldots, \tau_k(O_k)$. Note that the box-pleated hinge pattern is invariant under such transformations, so our crease patterns remain subsets of the box-pleat grid.

To optimize one of the metrics defined above, we use a brute-force algorithm which tries all orientations of all k crease patterns, and for each, evaluate the metric by constructing the multiple origami plan as described in previous sections. Our overall planner chooses the orientations that achieve the best value for the desired metric.

Table III (right half) shows some measured running times of our implementation. Table 6 shows the range of the three different objectives for an example integrating three single origami plans. To show the impact of this optimization procedure, we also compute the *worst* choice for each metric, which is what an arbitrary choice of orientations might give. Results for other examples are similar in overall behavior. We observe that the three objectives tend to improve together, as they all aim to have fortuitous overlaps between the single origami plans. Figure 23 shows the resulting group decompositions for the best orientation for each of the three objectives.

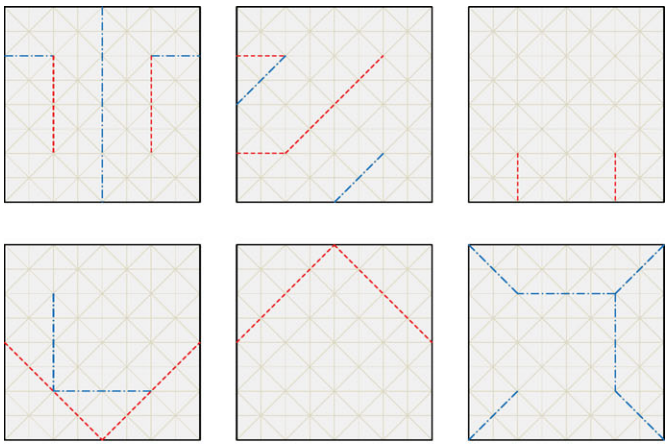
7. Conclusions

We have described a suite of algorithms for planning the creation of objects using self-folding sheets. We described the algorithm for automatically planning the creation of one object. We also presented an algorithm for the automatic creation of multiple objects from a single sheet. Finally we described design optimization issues by presenting a method

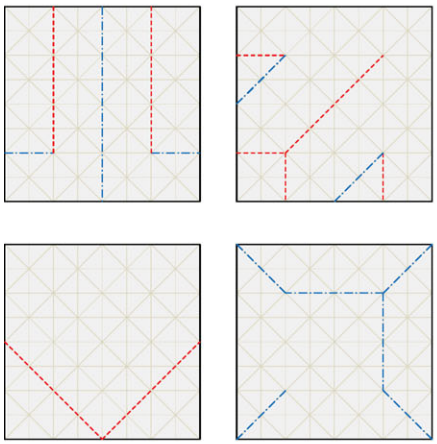
https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

Table V. Results from optimizing (minimizing) and anti-optimizing (maximizing) the numbers of actuators, groups, and bidirectional edges for a multiple origami plan of the airplane, boat, and table. The second and third rows divide the first row by the total number of edge actuators among the three single origami plans, and by the number of hinges in an 8 × 8 box-pleat pattern, respectively.

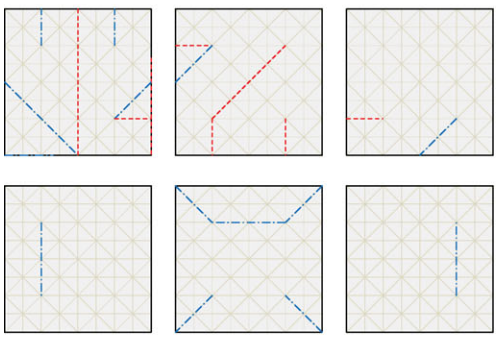
	Optimization (minimization)			Extreme worst (maximization)		
Metric	#actuators	#groups	#biedges	#actuators	#groups	#biedges
#edge actuators	76	80	76	92	86	88
%all edges in single origami plans	73.1%	76.9%	73.1%	88.4%	82.7%	84.7%
%all edges in 8 × 8 box-pleat	12.2%	12.8%	12.2%	14.7%	13.8%	14.1%
#groups	6	5	7	5	7	6
#bidirectional edges	14	14	10	26	24	26



(a) Minimizing number of actuators.



(b) Minimizing number of groups.



(c) Minimizing number of bidirectional edges.

Fig. 23. Group decompositions resulting from the optimized multiple-origami planner, with the three different objectives.

for automatically planning the threading of a self-folding sheet so that one actuator can drive multiple creases of the object. These algorithms are designed for sheets with built-in creases and embedded actuators, sensors, and connectors. The algorithms are assumed to run off-board the robot in the current form. Embedded computation and communication to the self-folding sheets will enable the algorithms to run on-board the sheet in a possibly distributed way, where each tile can be viewed as a module with a dedicated processor.

Much work remains to be done in order to develop a complete theory of self-folding sheets. The main challenge is to characterize which box-pleated origami designs can be folded rigidly, or at least design a wide family for which this is guaranteed to be possible. Ideally, this would also allow us to replace our continuous planner with one that guarantees a successful continuous folding. Changing the geometry of the hinge pattern may also help; in particular, we believe that adding slits to the sheet in a regular pattern may enable rigid folding of all polycubes.

The algorithms described in this paper are centralized and computed off-board. This approach is appropriate for the designing sheets that are specific to making a few different shapes, which is where the technology is currently most practical. In the future, however, we plan to build universal sheets where every edge has an actuator, enabling the possibility of online planning for new shapes. For this to become reality, an important next step is the development of a decentralized algorithm for multiple origami planning that can be computed on-board the robot. Another issue is how the user can interact with the sheet in the field, e.g., to select the required shape, when a computer may not be available.

Acknowledgments

Support for this work has been provided in part by the DARPA Programmable Matter project under contract W911NF-08-1-0228. We are grateful for this support. We are also grateful to the team lead by Prof. Robert Wood at Harvard, the team lead by Profs. Ron Fearing and Ali Javey at U. C. Berkeley, and the team lead by Profs. Vijay Kumar and Mark Yim at U. Pennsylvania for very exciting discussions and collaborations during the course of this work. We also thank Prof. Sangbae Kim and Martin Demaine for invaluable discussions and insights. We thank Prof. Tomohiro Tachi for providing a modification to his Rigid Origami Simulator to enable our continuous unfolding of folded states. Finally we thank the anonymous referees for their many helpful comments on the presentation of this paper.

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press

References

1. B. An, "Em-Cube: Cube-Shaped, Self-Reconfigurable Robots Sliding on Structure Surfaces," *Proceedings of the IEEE International Conference on Robotics and Automation*, New York (May 2008) pp. 3149–3155.
2. B. Aspvall, M. F. Plass and R. E. Tarjan, "A linear-time algorithm for testing the truth of certain quantified boolean formulas," *Inf. Process. Lett.* **8**(3), 121–123 (1979).
3. D. Balkcom, Robotic Origami Folding *PhD Thesis* (Pittsburgh, PA: Robotics Institute, Carnegie Mellon University, Aug. 2004).
4. N. Benbernou, E. D. Demaine, M. L. Demaine and A. Ovadya, "A universal crease pattern for folding orthogonal shapes," arXiv:0909.5388 (Sep. 2009). http://arXiv.org/abs/0909.5388.
5. Z. Butler, R. Fitch and D. Rus, "Distributed control for unit-compressible robots: Goal-recogition, locomotion and splitting," *IEEE/ASME Trans. Mechatronics* **7**(4), 418–30 (2002).
6. Z. Butler, K. Kotay, D. Rus and K. Tomita, "Generic decentralized control for lattice-based self-reconfigurable robots," *Int. J. Robot. Res.* **23**(9), 919–937 (2004).
7. D. J. Balkcom and M. T. Mason, "Introducing Robotic Origami Folding," *IEEE International Conference on Robotics and Automation* (2004), pp. 3245–3250.
8. D. Balkcom and M. Mason, "Robotic origami folding," *Int. J. Robot. Res.* **27**(5), 613–627 (May 2008).
9. Z. J. Butler and D. Rus, "Distributed planning and control for modular robots with unit-compressible modules," *Int. J. Robot. Res.* **22**(9), 699–716 (2003).
10. C.-H. Chiang and G. Chirikjian, "Modular robot motion planning using similarity metrics," *Auton. Robots* **10**(1), 91–106 (2001).
11. J. H. Cantarella, E. D. Demaine, H. N. Iben and J. F. O'Brien, "An Energy-Driven Approach to Linkage Unfolding," *Proceedings of the 20th Annual ACM Symposium on Computational Geometry* Brooklyn, New York (June 2004) pp. 134–143.
12. J. M. Díaz-Báñez and J. A. Mesa, "Fitting rectilinear polygonal curves to a set of points in the plane," *Eur. J. Oper. Res.* **130**(1), 214–222 (2001).
13. J. S. Dai and D. G. Caldwell, "Origami-based robotic paper-and-board packaging for food industry," *Trends Food Sci. Technol.* **21**(3), 153–157 (2010).
14. E. D. Demaine, S. L. Devadoss, J. S. B. Mitchell and J. O'Rourke, "Continuous Foldability of Polygonal Paper," *Proceedings of the 16th Canadian Conference on Computational Geometry* Montréal, Canada (Aug. 2004), pp. 64–67.
15. R. G. Downey and M. R. Fellows, *Parameterized Complexity* (Springer-Verlag, New York 1999).
16. E. D. Demaine and J. O'Rourke, *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*, (Cambridge University Press, Cambridge, July 2007).
17. C. Detweiler, M. Vona, Y. Yoon, S. Yun and D. Rus, "Self-assembling mobile linkages with active and passive modules," *IEEE Robot. Autom. Mag.* **14**(4), 45–55 (2007).
18. T. Fukuda and Y. Kawakuchi, "Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator," *Proceedings of IEEE International Conference on Robotics and Automation* (1990) pp. 662–667.
19. K. Gilpin, K. Kotay, D. Rus and I. Vasilescu, "Miche: Modular shape formation by self-disassembly," *Int. J. Robot. Res.* **27**(3–4), 345–372 (2008).
20. K. Gilpin, K. Kotay, D. Rus and I. Vasilescu, "Miche: Self-assembly by self-disassembly," *Int. J. Robot. Res.* **27**(3–4), 345–372 (2008).
21. E. Hawkes, B. K. An, N. M. Benbernou, H. Tanaka, S. Kim, E. D. Demaine, D. Rus and R. J. Wood, "Programmable matter by folding," *Proc. Natl. Acad. Sci. U. S. A.* **107**(28), 12441–12445 (2010).
22. P. J. White, K. Kopanski and H. Lipson, "Stochastic self-reconfigurable cellular robotics," *Proceedings of the IEEE International Conference on Robotics and Automation*, New Orleans, LA (May 2004) pp. 2888–2893.
23. E. Klavins, S. Burden and N. Napp, "Optimal rules for programmed stochastic self-assembly," *Robotics: Science and Systems*, Philadelphia, PA (2006).
24. L. E. Kavraki, P. Svestka, J.-C. Latombe and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.* **12**(4), 566–580 (1996).
25. L. Lu and S. Akella, "Folding cartons with fixtures: A motion planning approach," *IEEE Trans. Robot. Autom.* **16**(4), 346–356 (2000).
26. H. Lipson and J. Pollack, "Automatic design and manufacture of robotic lifeforms," *Nature* **406**, 974–978 (2000).
27. S. Murata, E. Yoshida, K. Tomita, H. Kurokawa, A. Kamimura and S. Kokaji, "Hardware Design of Modular Robotic System," *Proceedings of the International Conference on Intelligent Robots and Systems*, Takamatsu, Japan (2000) pp. 2210–2217.
28. R. Nagpal, Programmable Self-assembly: Constructing Global Shape Using Biologically-Inspired Local Interactions and Origami Mathematics *PhD Thesis* (Massachusetts Institute of Technology, 2001).
29. R. Nagpal, "Self-Assembling Global Shape, Using Ideas From Biology and Origami," *Origami^3: Third International Meeting of Origami Science, Math and Education* (T. Hull, ed.) (A K Peters, 2002) pp. 219–231.
30. A. Pamecha, I. Ebert-Uphoff and G. Chirikjian, "Useful metrics for modular robot motion planning," *IEEE Trans. Robot. Autom.* **13**(4), 531–45 (1997).
31. G. Song and N. M. Amato, "A motion-planning approach to folding: from paper craft to protein folding," *IEEE Trans. Robot. Autom.* **20**(1), 60–71 (Feb. 2004).
32. W.-M. Shen, M. Krivokon, H. Chiu, J. Everist, M. Rubenstein and J. Venkatesh, "Multimode locomotion for reconfigurable robots," *Auton. Robots* **20**(2), 165–177 (2006).
33. S.-M. Belcastro and T. C. Hull, "A Mathematical Model for Non-Flat Origami," *Origami^3: Proceedings of the 3rd International Meeting of Origami Mathematics, Science and Education* (2002) pp. 39–51.
34. T. Tachi, "Simulation of Rigid Origami," *Origami^4: Proceedings of the 4th International Meeting of Origami Science, Math and Education* (2009), pp. 175–187.
35. P. Varshavskaya, L. P. Kaelbling and D. Rus, "Automated design of adaptive controllers for modular robots using reinforcement learning," *Int. J. Robot. Res. Special Issue on Self-Reconfigurable Modular Robots* (2007).
36. M. Yim, W-M. Shen, B. Salemi, D. Rus, H. Lipson, E. Klavins and G. Chirikjian, "Modular self-reconfiguring robot systems: Opportunities and challenges," *IEEE/ASME Trans. Mechatronics* (2006).
37. M. Yim, Y. Zhang, J. Lamping and E. Mao, "Distributed control for 3D shape metamorphosis," *Auton. Robots* **10**(1), 41–56 (2001).

https://doi.org/10.1017/S0263574710000731 Published online by Cambridge University Press