Integer programming models for flat origami

Goran Konjevod

1 Introduction

Most traditional and many contemporary origami models fold flat, or at least have flat-foldable crease patterns. Thus, it would be very useful to have a complete mathematical description of flat-foldable origami. This has often been stated as a major open problem in the mathematical foundations of origami. There are three main mathematical properties of an origami: continuity, piecewise isometry, and noncrossing. Justin [7] proposed a set of noncrossing axioms and claimed their validity was not only necessary but sufficient for flat-foldability. His proof, however is not very formal and the problem of whether they are sufficient has remained open. Recently, E. Demaine [3] announced a positive answer to this question.

However, a mathematical description is of limited practical use unless it is effective, that is, unless it comes with (preferably efficient) procedures for practical manipulation of the objects it describes.

In case of flat-foldability, there are several types of questions an effective model should be able to answer. The simplest is that of deciding flat foldability of a crease pattern[1]: given a set of creases, with their orientations (mountain or valley) assigned, is there a flat origami with exactly the given creases taking on exactly the given orientations? Even this problem is unlikely to have an efficient algorithm because it is NP-complete, as shown by Bern and Hayes [1].

More general is the *design problem*: given a certain property required of a flat origami, for example a given shape, or arrangement of flaps, or, for duo-colored paper, a color-change pattern, is it possible to design such a flat

[1]In this paper, a *crease pattern* includes not only the location of every crease in the model to be folded, but also the information on which creases are mountain and which valley folds.

fold? Ideally, in the case of a positive answer, the solution should also include a(n efficient) procedure to determine the crease pattern and the arrangement of layers in such a model.

What makes these problems particularly difficult at the current state of mathematical knowledge is that they are at the same time continuous and discrete. There is a continuum of possible creases, and so it is not clear how to model the problem using combinatorial techniques, which have been developed for working with finite sets, and yet at the heart of the foldability problem lies the combinatorial issue of arranging the layers correctly [1].

In order to bring the problem closer to what mathematics can currently deal with, we restrict it to a special case, only considering folds in which all the creases are either vertical, horizontal, or at a ± 45 degree angle, and each crease goes through a point of a fixed square grid (see Figure 1 for an example). Even though the problem now becomes discrete, and may appear simpler, the NP-completeness of flat-foldability remains.

We represent a flat origami by an *integer linear program*[2] [10] (ILP). The variables of the ILP will model both the creases and the arrangement of layers. If we then set the crease variables to predefined values, we get a specific instance of the ILP whose feasible solutions represent exactly the ways to flat fold the crease pattern. Thus flat-foldability of a crease pattern reduces to deciding if the corresponding ILP instance has any solutions. We will also show how certain properties (e.g. color-change) of an origami model can be described by linear constraints. In this case, defining additional variables and constraints, and setting them to desired values gives us an ILP instance whose solutions will provide the necessary values for crease variables and thus an answer to the design problem.

2 Integer programming model

2.1 Grids and creases

Original grid. We begin with a k by k square grid placed on the uncreased square sheet of paper, and allow folding only on the segments of the grid and the segments of diagonals of the *grid squares* (Figure 1). Additionally, we

[2]The word *program* is here used in the older sense of *planning*, as in *mathematical programming*, and not in the sense of computer programming.

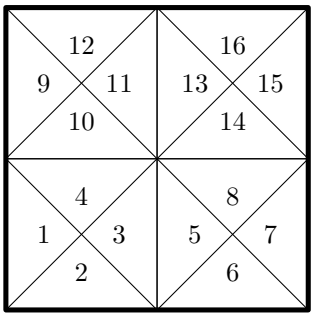

Figure 1: The 2 by 2 square triangle grid.

allow at most one of the diagonals of each grid square to be folded (this simplifies the model, and leads to no loss of generality).

The two diagonals in each grid square define four *grid triangles*. Due to the restrictions on where creases lie, every grid triangle will be flat and uncreased in the final folded state. We call these triangles the *basic polygons*. Some parts of the ILP model described here depend directly on the fact that the basic polygons are the grid triangles, but other parts of the model would still work even if this were not the case. We use V to denote the set of all basic polygons, and n for the number of elements in V, that is, $|V| = n = 4k^2$.

Creases. For each pair of adjacent basic polygons, the segment separating them may be a crease line. We define a *graph* based on this property, taking V as the set of vertices and adding an edge between two vertices if the basic polygons corresponding to these vertices are adjacent and separated by a potential crease. We label the set of all edges by E.

Folded model. Just like we named the elements of the unfolded sheet of paper by considering the grid of allowed creases, we must name the elements of the folded model. It is not difficult to see that as long as we fold only on the allowed creases, every basic polygon is folded into the location of another basic polygon, possibly lying in a grid that extends beyond the boundaries of the original sheet. This property simplifies the model, and is the main reason to work with this particular triangle-square grid.

We will in general know the area covered by the folded model. For example, if we are interested in figuring out how to fold something, the shape of the folded model is naturally given as a parameter of the problem. On the

other hand, if we are testing the foldability of a crease pattern, we can easily infer the location of each basic polygon from the crease pattern. (Indeed, first we fix one basic polygon to be at a location of our choice in the folded model. Everything else is done relative to this polygon. Then for each other basic polygon we can trace a path along the grid segments from the one fixed polygon, and account for each crease by changing the direction of the path appropriately, until we reach the polygon of interest.)

Thus we name the set of locations where original basic polygons may lie in the folded model: W, and the set of adjacent pairs of polygons in the folded model: F.

2.2 Crease variables and constraints

In order to have a set of numbers describe a fold, we relate them to the features of the fold. The easiest to understand are the crease variables. For each crease segment $e \in E$, we have two variables, f_e^v and f_e^m. In a fold where the segment e is (a part of) a valley fold, we have $f_e^v = 1$. In the case of a mountain fold, $f_e^m = 1$. If the segment e is not folded, then $f_e^v = f_e^m = 0$.

It is clear that the inequality $f_e^v + f_e^m \leq 1$ is always satisfied: if the segment e is folded at all, it is either a mountain or a valley fold, but not both. For each crease segment e we have such an inequality in our model.

2.3 Orientation and location constraints

If we are to describe a folded model, we need to know exactly where every basic polygon is located after folding. Folding along a segment flips a part of the sheet about the segment as the axis, turning the folded part of the sheet upside-down and changing the location of every one of its basic polygons. By examining each crease segment and checking whether it is folded, we can figure out exactly where every basic polygon ends up after folding. To describe the location of each polygon, we use *location assignment* variables. Given a basic polygon $v \in V$ of the unfolded grid and a basic polygon $w \in W$ of the folded model, we define $x(v, w)$ to be 1 if v ends up at the same location as w, and 0 otherwise. Since there are no cuts, every basic polygon of the unfolded grid is still present in the folded model, and so it must be true that $\sum_{w \in W} x(v, w) = 1$ for every $v \in V$ (that is, every basic polygon is mapped somewhere).

This is not enough to describe the folded model, however. The basic polygons are isosceles right triangles, and thus symmetric about the line passing through their right angle vertex and dividing the hypothenuse in half. Therefore, a basic polygon can be mapped to the same location in two ways: its original top side may still be at the top after folding, or not. To capture this, we use the variable σ_v for every $v \in V$. If the original top side of v is still its top side, then $\sigma_v = 0$ (otherwise, $\sigma_v = 1$). We refer to σ_v also as the *orientation* of v. Clearly for every v, either $\sigma_v = 1$ or $\sigma_v = 0$.

To understand the following sets of constraints, consider a grid segment e. The orientations of its two defining polygons u and v depend on whether e is folded or not. The next four constraints describe completely the relation between the orientations σ_u, σ_v and the fold variables $f^m(u,v), f^v(u,v)$:

$$\begin{aligned}
\sigma_u - \sigma_v &\leq f^m(u,v) + f^v(u,v) \\
\sigma_v - \sigma_u &\leq f^m(u,v) + f^v(u,v) \\
\sigma_u + \sigma_v &\geq f^m(u,v) + f^v(u,v) \\
\sigma_u + \sigma_v &\leq 2 - f^m(u,v) - f^v(u,v).
\end{aligned}$$

Location is a little more difficult to characterize. However, the location of a basic polygon is still determined completely by the location of any one of its neighbors and by the value of their common crease variable. Say $v, v' \in V$ are two neighboring basic polygons, sharing the grid segment e. Suppose v' is mapped to the basic polygon w of the folded model, that is, $x(v', w) = 1$. Then stating that v is mapped to w is equivalent to stating that e is folded, in other words, $x(v, w) = f^v(v, v') + f^m(v, v')$. Here we have a condition that says "$x(v', w) = 1$ if and only if this equation holds". Our goal is, of course, to write all this as a linear equation. The key is to notice that if $x(v', w) = 0$, our constraint should require nothing. Here's how to do this:

$$x(v, w) \geq x(v', w) + f^v(v, v') + f^m(v, v') - 1.$$

In other words, if both $x(v', w)$ and one of the fold variables are set to 1, then $x(v, w)$ will be forced to 1 as well. If one of the former is 0, then the constraint will simply say $x(v, w) \geq 0$, which is true anyway.

Of course, it is possible that the segment shared by v and v' is not folded. In that case, v and v' will not be mapped to the same fold polygon, but to adjacent ones. Which ones exactly, will depend on their orientation. The constraints are somewhat more complicated for this case, and we will not explain the details here.

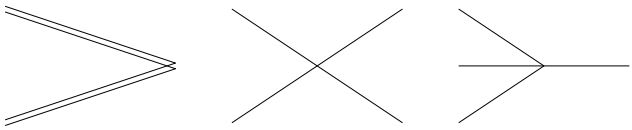

Figure 2: The three crossing types.

2.4 Non-crossing constraints

The constraints described so far ensure that the locations of points in the fold satisfy two of the three basic properties of flat origami: continuity and isometry (no ripping or stretching of paper). The noncrossing property is the one that actually makes the problem difficult, and will also cause us the most trouble.

Assuming the Justin axioms (see Introduction for background), there are three types of crossings that a flat-foldable origami avoids, as illustrated in Figure 2: we refer to them by mnemonics W, X and Y.

In our simplified version, where all folds lie along the square-triangle grid, any imaginable crossing would happen along a grid segment as well, so as with orientation and location, it suffices to enforce non-crossing for pairs of adjacent basic polygons in the grid of the folded model. Consider a flat-folded model. At any point, there may be several layers of paper one below the other. Number them from the bottom, starting from 1. Do this independently for each basic polygon of the fold grid, and we will have for every polygon in the fold an ordering of all the polygons mapped to it by the fold. We next show how to force the integer program to assign layers to basic polygons in the, and then it will be clear how to complete the constraint set, because it is not difficult to express the non-crossing constraints in terms of layers. (The same idea is used by Jonathan Schneider [9] who calls it *superposition ordering* in order to describe properties of flat-foldable crease patterns.)

For example, suppose w and w' are two neighboring polygons in the fold. Say two neighboring basic polygons, v and v', are mapped to w. They share a grid segment e. Suppose f is the segment of the fold grid to which e is mapped and w' the fold polygon adjacent to w but on the other side of f. If another basic polygon z is mapped to w and lies between v and v' in the fold (that is, it is at a layer between the layers of v and v'), then there will be a non-crossing constraint of type Y that will say that it is impossible for a neighbor of z to be mapped to w'.

Layering constraints. We first make sure that every basic grid polygon is assigned to a layer:

$$\sum_{k \in L} \lambda(v, w, k) = x(v, w),$$

for all $v \in V$ and $w \in W$, where L is the set of all possible layers (that is, the set of numbers $\{1, 2, \ldots, L_{\max}\}$ for some large enough $L_{\max}$). What the constraint really says is that if v is mapped to w by the fold, then it lies at some layer over w. The value of the variable $\lambda(v, w, k)$ is 1 if v lies at layer k over the fold polygon w, and 0 otherwise.

Then we make sure that layers fill up starting from the bottom by requiring that if layer k is empty, then so are all the layers above it:

$$\sum_{v \in V} \lambda(v, w, k) \leq \sum_{v \in V} \lambda(v, w, k-1),$$

for all $w \in W$ and $k > 1$.

Finally, we make sure that at each layer there is at most one polygon:

$$\sum_{v \in V} \lambda(v, w, k) \leq 1,$$

for all k and all $w \in W$.

Now we can compute the layer at which a polygon v lies. First, let $l(v, k) = 1$ if v is at layer k, and $l(v, k) = 0$ otherwise. Then we have

$$\begin{aligned} \lambda(v, w, k) &\leq l(v, k) \\ \lambda(v, w, k) &\geq x(v, w) + l(v, k) - 1. \end{aligned}$$

In order to write constraints that compare layers of different basic polygons, we use the variable ll_v to be the exact layer of the basic polygon v. This value can be expressed directly in terms of l:

$$ll_v = \sum_{k \in L} k \cdot l(v, k).$$

In comparing the layers we do not care about their values, only about the sign of their difference. We use $\alpha(u, v)$ to denote whether the polygon u lies above polygon v (that is, whether $ll_u > ll_v$). First, no polygon can lie above itself:

$$\alpha(v, v) = 0,$$

for all $v \in V$. Then, if u is above v then v cannot be above u:

$$\alpha(u, v) + \alpha(v, u) \leq 1,$$

for all $u, v \in V$. Finally, we define α in terms of ll:

$$\alpha(u, v) \geq \frac{ll_u - ll_v}{L_{\max}}.$$

This suffices to establish an ordering among the polygons mapped to the same location, but additional constraints may be useful. What happens when the model is used to solve a problem is that a complicated algorithm examines many possibilities for assigning the 0-1 values to the variables of the model, and attempts to eliminate inconsistent assignments as efficiently as possible. Additional constraints usually help in such a situation, and therefore in the actual implementation of the model we also enforce other constraints that relate the ordering constraints to orientation and location constraints.

3 Example and conclusion

Due to limited space, we do not enumerate all the constraints. The complete integer programming model (written in the modeling language AMPL [5]) can be found on the author's web page.

The appendix is a short example showing how the model can be used. The simplest approach is to list additional constraints that are to be enforced in order to model an actual fold. The given example gives the crease pattern illustrated in Figure 3 of the iso-area 2 by 2 chessboard folded from a 4 by 4 square.

This example is very simple, however the current version of the model results in a very large integer program even for this small crease pattern, and takes several hours to solve on a reasonably fast computer (an AMD64-based machine). The model can undoubtedly be improved and made more tractable. This is usually done by a careful examination of constraints. Some types of linear constraints (such as the one we used to define a lower bound on $\alpha(u, v)$) are computationally awkward, in that they cause the integer program solver to generate too many cases that all have to be solved. It doesn't seem obvious how to replace these constraints by better ones, but there are very likely additional inequalities that will reduce the search space. Schneider

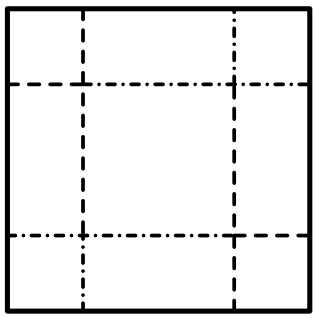

Figure 3: The 2 by 2 iso-area chessboard crease pattern.

describes several necessary conditions for flat foldability, some of which do not follow immediately from our location and orientation. It may be possible to derive further inequalities based on these conditions and thus make the solver's job easier. A simple improvement for the foldability problem (but not for the design problem) would be to precompute all the point locations and then have the integer program "only" test if there is a valid layering.

This work was motivated by attempts to prove bounds on the size of a k by k chessboard that can be folded out of a unit square of black-and-white paper. Hulme's chessboard [6] was the first one, giving an 8 by 8 board from a 64 by 64 square, with a "reduction factor" of 8. Montroll's board [8] uses a 36 by 36 square, and those of Chen [2] and Dureisseix [4] 32 by 32, for a reduction factor of 4. It is conjectured that the latter two are optimal, that is, that an 8 by 8 board with correctly colored squares cannot be folded from a square smaller than 32 by 32. (In general, according to this conjecture, a k by k board would require a reduction factor of $2k$ in the even case.) For now, our work leaves this question open. As far as we can tell, even for folding a 4×4 chessboard, there are no rigorous proofs that our current best designs (out of 8×8 square grid, or 10×10 for a seamless design) are optimal.

Appendix: additional constraints for the 2 by 2 iso-area board

```
set Valley within E:={(11,13), (16,30), (19,21), (27,29),
  (35,37), (36,50), (43,45), (51,53)};

set Mountain within E:={(3,5), (4,18), (8,22), (12,26),
  (40,54), (44,58), (48,62), (59,61)};

subject to valleys{(a,b) in E: (a,b) in Valley or (b,a) in Valley}:
  fv[a,b] = 1;

subject to mountains{(a,b) in E: (a,b) in Mountain or
  (b,a) in Mountain}:  fm[a,b] = 1;

subject to ss: s[1] = 1;

subject to flat{(a,b) in E: not((a,b) in Valley union Mountain)
 and not((b,a) in Valley union Mountain)}: fv[a,b] + fm[a,b] = 0;
```

References

[1] M. Bern and B. Hayes. The complexity of flat origami. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 175–183, 1996.

[2] S. Chen. Checkerboard. In *Proceedings of the Annual OUSA Convention*, pages 72–75, 2001.

[3] E. D. Demaine. Personal communication, 2006.

[4] D. Dureisseix. Chessboard. Unpublished diagram, 2001.

[5] R. Fourer, D. M. Gay, and B. W. Kernighan. A modeling language for mathematical programming. *Management Science*, 36:519–554, 1990.

[6] M. Hulme. *BOS Booklet 7: Chess Sets*. BOS, 1985.

[7] J. Justin. Towards a mathematical theory of origami. In *Origami Science and Art: Proceedings of the Second International Meeting of Origami and Scientific Origami*, pages 15–29, 1997.

[8] J. Montroll. *Origami Inside-Out*. Dover, 1993.

[9] J. Schneider. Flat-foldability of origami crease patterns. Manuscript, 2005.

[10] A. Schrijver. *Theory of integer and linear programming*. Wiley, 1986.