Title	Flat Foldings of Plane Graphs with Prescribed Angles and Edge
Author(s)	Abel, Zachary; Demaine, Erik D.; Demaine, Martin; Eppstein, David; Lubiw, Anna; Ryuhei Uehara
Citation	Lecture Notes in Computer Science, 8871: 272-283
Issue Date	2014-09-24
Type	Journal Article
Text version	author
URL	http://hdl.handle.net/10119/13761
Rights	This is the author-created version of Springer, Zachary Abel, Erik D. Demaine, Martin Demaine, David Eppstein, Anna Lubiw and Ryuhei Uehara, Lecture Notes in Computer Science, 8871, 2014, 272-283. The original publication is available at www.springerlink.com, http://dx.doi.org/10.1007/978-3-662-45803-7_23
Description	Graph Drawing, 22nd International Symposium, GD 2014, Würzburg, Germany, September 24-26, 2014, Revised Selected Papers

Flat Foldings of Plane Graphs with Prescribed Angles and Edge Lengths

Zachary Abel[1], Erik D. Demaine[2], Martin L. Demaine[2], David Eppstein[3], Anna Lubiw[4], and Ryuhei Uehara[5]

[1] Department of Mathematics, MIT, Cambridge, USA
[2] MIT Computer Science and Artificial Intelligence Lab., Cambridge, USA
[3] Department of Computer Science, University of California, Irvine, USA
[4] David R. Cheriton School of Computer Science, University of Waterloo, Canada
[5] School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa, Japan

Abstract. When can a plane graph with prescribed edge lengths and prescribed angles (from among $\{0, 180°, 360°\}$) be folded flat to lie in an infinitesimally thick line, without crossings? This problem generalizes the classic theory of single-vertex flat origami with prescribed mountain-valley assignment, which corresponds to the case of a cycle graph. We characterize such flat-foldable plane graphs by two obviously necessary but also sufficient conditions, proving a conjecture made in 2001: the angles at each vertex should sum to $360°$, and every face of the graph must itself be flat foldable. This characterization leads to a linear-time algorithm for testing flat foldability of plane graphs with prescribed edge lengths and angles, and a polynomial-time algorithm for counting the number of distinct folded states.

1 Introduction

The modern field of origami mathematics began in the late 1980s with the goal of characterizing flat-foldable crease patterns, i.e., which plane graphs form the crease lines in a flat folding of a piece of paper [12]. This problem turns out to be NP-complete in the general case, with or without an assignment of which folds are mountains and which are valleys [6].

On the other hand, flat foldability can be solved in polynomial time for crease patterns with just a single vertex (thus characterizing the local behavior of a vertex in a larger graph). By slicing the paper with a small sphere centered at the single vertex (the geometric *link* of the vertex), single-vertex flat foldability reduces to the 1D problem of folding a polygon (closed polygonal chain) onto a line; see Figure 1. This problem can be solved by a greedy algorithm that repeatedly folds both ends of a shortest edge with opposite fold directions (mountain and valley)—either because such directions have already been pre-assigned or, if the mountain-valley assignment is not given, by making such an assignment [4, 6, 12, 20]. The spherical, self-touching Carpenter's Rule Theorem [1, 11, 26] implies that any flat-folded single-vertex origami can be reached
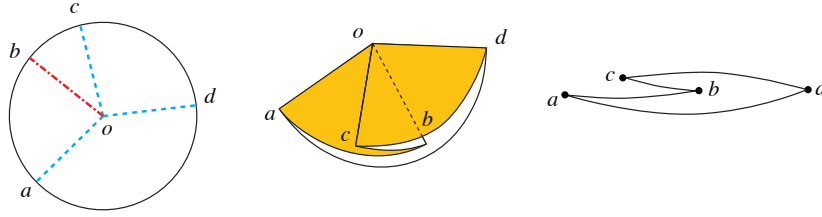
Fig. 1. Flat folding at a single vertex on a disc reduces to the problem of folding a polygon onto a line.
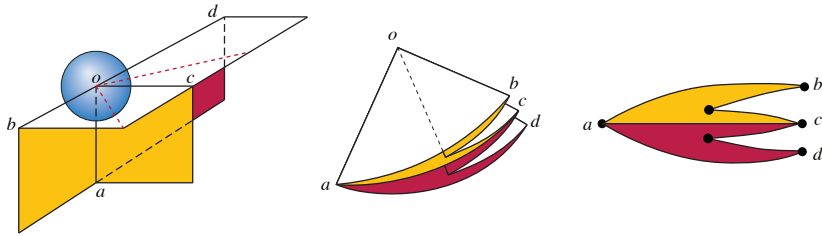

Fig. 2. Flat folding a two-dimensional cell complex with a single vertex reduces to the problem of folding a plane graph onto a line.

from the unfolded piece of paper by a continuous motion that avoids bending or folding the uncreased parts of the paper.

In practical applications of folding beyond origami, the object being folded may not be a single flat sheet, but rather some 2D polyhedral cell complex with nonmanifold topology (more than two facets joined at an edge). Flat foldability of such complexes is only harder than the origami case, but again we can look at the case of a complex with a single vertex. As above, we can reduce the problem to 1D by slicing with a small sphere centered at the vertex—now resulting in a general plane graph rather than a simple cycle—and asking whether this graph can be flattened onto a line [2]; see Figure 2. In this way, the problem of flat-folding single-vertex complexes can be reduced to finding embeddings of a given plane graph onto a line.

It is this problem that we study here: given a plane graph with specified edge lengths, does it have a straight-line plane embedding with all vertices arbitrarily close to a given line and with all edges arbitrarily close to their specified lengths? In the version of the problem we study, we are additionally given a specification of whether the angle between every two consecutive edges at each vertex is a *mountain fold* (the angle is arbitrarily close to 0), a *valley fold* (the angle is arbitrarily close to 360°), or *flat* (the angle is arbitrarily close to 180°). Without this information, the problem of testing whether a given plane graph can be folded flat with specified edge lengths (allowing angles of 180°) is weakly NP-complete, even for graphs that are just simple cycles, by a straightforward reduction from

	Flat angles forbidden	Flat angles allowed
Angle assignment given	Linear time (new)	Linear time (new)
Angle assignment unspecified	Open	NP-complete [2]

Table 1. Complexity of flat folding a plane graph, by input model

the subset sum problem. For general plane graphs, the problem becomes strongly NP-complete [2]. Therefore, we concentrate in this paper on the version of the problem with given angle assignments, posed as an open problem in [2].

1.1 New Results

We show that it is possible to test in linear time whether a given plane graph, with given edge lengths and angle assignment, can be folded flat; refer to Table 1. Additionally, in polynomial time, we can count the number of combinatorial distinct flat foldings.

Our algorithms are based on a new characterization of flat-foldable graphs: a flat folding exists if and only if the angles at each vertex sum to 360° and each individual face in the given graph can be folded flat. Even stronger, we show that independent flat foldings of the interior of each face can always be combined into a flat folding of the whole graph. Figure 3 shows an example of this combination of face foldings. A form of the theorem was conjectured in 2001 by Ilya Baran, Erik Demaine, and Martin Demaine, but not proved until now; it contradicts the intuitive (but false) idea that, for faces with ambiguous spiraling shapes, the face must be folded consistently with its neighboring faces. With this theorem in hand, our algorithms for constructing and counting folded states follow by using a greedy "crimping" strategy for flat-folding simple cycles [4,6,12] and by using dynamic programming to count cycles within each face.

Our characterization necessarily concerns flat folded states, not continuous folding motions from a given (nonflat) configuration. As shown by past work, even for trees, there exist locked states that cannot be continuously moved to a flat folded state [5,8], and testing the existence of a continuous motion between two states is PSPACE-complete [3].

We leave open the problem of finding a flat folded state for a graph in which the planar embedding and edge lengths are preassigned, and angles of 180° are forbidden, but the choice of which angles at each vertex are 0 and which are 360° is left free (bottom-left cell of Table 1). Even for trees, this open problem appears to be nontrivial; see Figure 4 and [15].

1.2 Related Work

There has been intensive study of straight-line drawings of graphs with specified edge lengths and/or specified angles between consecutive edges in a cyclic ordering of edges around each vertex. If only edge lengths are specified then—whether
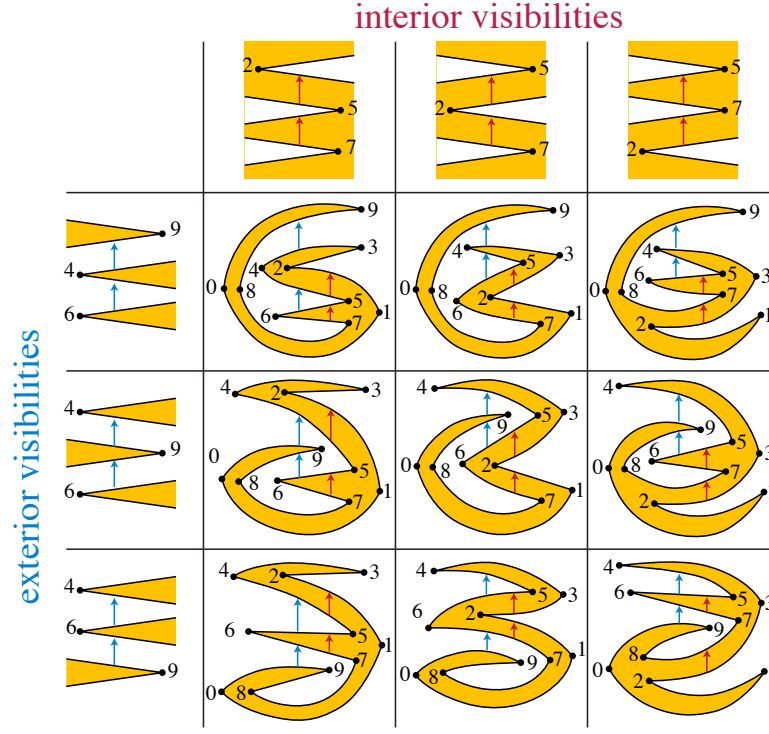
Fig. 3. A planar graph with two faces, each of which can be flat-folded to give three different patterns of vertical visibility (or "touching") within it. These patterns can be combined independently, giving nine flat-foldings of the whole graph.

the drawing must be planar or not—the problem is NP-hard [9,24], or worse [25]. It is also NP-hard to draw a plane graph with specified angles [16]. If both edge lengths and angles are specified then the drawing is uniquely determined and easy to construct, except in situations like ours where coincident edges give rise to ambiguities.

There are a number of results for special cases that have a similar flavor to ours, in that the whole plane graph can be realized if and only if each face can. Most of these cases arise as the prelude to finding an appropriate angle assignment.

Upward Planarity. A directed acyclic graph (DAG) is *upward planar* [17] if it has a planar drawing in which each edge is drawn as an increasing *y*-monotone curve. Recognizing upward planar graphs is NP-hard [18] but Bertolazzi et al. [7] gave a polynomial time algorithm for the special case of a plane graph whose cyclic order of edges around each vertex is prescribed. The main issue in their solution is to distinguish "small" versus "large" angles; if an upward planar drawing is flattened onto a vertical line, then its small and large angles correspond
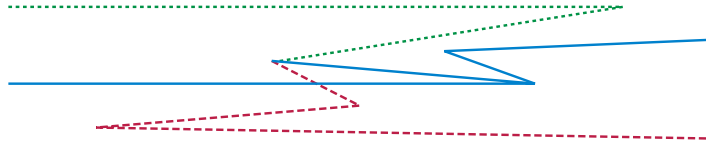
Fig. 4. A tree with fixed edge lengths that (when angles equal to 180° are forbidden) has no flat folding, regardless of planar embedding or angle assignment

to our valley and mountain folds. The angle assignment is forced except at vertices with only incoming or only outgoing edges, where exactly one angle should be large and the rest small. Bertolazzi et al. used network flows to determine these angles. To prove their algorithm's correctness, they showed that a graph with a given angle assignment has an upward planar drawing if and only if each face cycle has an upward planar drawing. The condition for drawing a single face, given an angle assignment, is particularly simple: an acyclic orientation of a cycle has an upward planar drawing if and only if it has two more small than large angles. Their proof also shows that embedding choices for the faces can be combined arbitrarily.

Level Planarity. Our flat folding problem differs from upward planarity in that we have assigned edge lengths as well as assigned angles. This makes it more similar to the problem of *level planarity* [13, 19, 21]. The input to this problem is a *leveled* directed acyclic graph: a DAG whose vertices have been partitioned into a sequence of levels (independent sets of its vertices), with all edges directed from earlier to later levels. The goal is to find an upward planar embedding that places the vertices of each level on a horizontal line [22]). This problem has a linear time solution [21] based on PQ-trees. When the cyclic order of edges around vertices is specified (and in fact for more general constraints) there is a quadratic time solution based on solving systems of binary equations [19].

The input to our folding problem may be interpreted as a leveled plane DAG. (Since our convention is to flatten to a horizontal line, we will map to a level planarity problem with levels progressing rightward rather than upward—this is a superficial difference.) Arbitrarily choose an x-coordinate for one vertex and a direction (left-to-right) for one edge incident to that vertex. These choices can be propagated to all the vertices and edges using the specified edge lengths
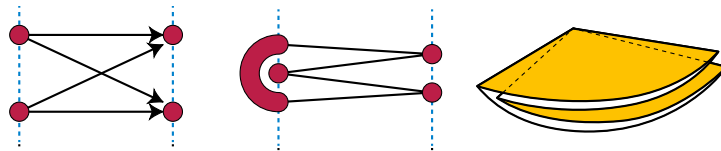

Fig. 5. A four-vertex cycle with vertices alternating between two levels is not level planar, but can be folded flat, representing a sheet of paper folded into quarters.

5

and angles. The set of vertices at a given x-coordinate constitute a level, giving us an input to the level planarity problem with a prescribed plane embedding. However, the embeddings we seek in the folding problem are not the same as level planar embeddings. In a level planar embedding, vertices within a single level must be linearly ordered by the second coordinate value. In contrast, in the folding problem a vertex that has only incoming or only outgoing edges may be nested between two adjacent edges of another vertex at the same level. A four-vertex cycle, oriented with alternating edge directions, illustrates the difference between these two types of embedding: it is not level planar, but it still has a flat folding with three mountain folds and one valley fold, corresponding to the usual way of folding a square sheet of paper into quarters (Figure 5).

We have therefore been unable to apply level planarity algorithms to solve our flattening problem. On the other hand, our algorithm can be used to test level planarity of a plane graph (with a linear order of incoming and outgoing edges at each vertex) in linear time. Given an input to the level planarity problem, we assign increasing coordinates to the levels, and assign the length of an edge to be the difference in coordinates between the levels of its endpoints. Mountain/valley/flat angles are determined from the level assignment. Finally, to preclude the nesting of vertices that is allowed in flattening but not in level planarity, we add an extra edge incident to each vertex that has only incoming or only outgoing edges: if vertex v on level i has only outgoing edges, we add a new incoming edge from a new level just before i. The resulting plane graph has a flat-folding respecting the angles and edge-lengths if and only if the original has a level planar drawing. From this we also obtain the result that a leveled plane graph has a level planar drawing if and only if each cycle does.

Rectilinear Planarity. In our flat folding problem, the angles are multiples of $180°$. When angles are multiples of $90°$, we arrive at the important problems of orthogonal and rectilinear graph drawing [14]. A graph is *rectilinear planar* if it can be drawn in the plane so that every edge is a horizontal or vertical line segment. Coincident edges are forbidden, so the graph must have maximum degree 4. This problem is NP-complete in general [18] but—as with upward planarity—there is a polynomial time algorithm, due to Tamassia [27], if the cyclic order of edges around vertices is prescribed. Again, the main issue is to find an assignment of angles or, equivalently, a labeling of the edges incident to a vertex with distinct labels from the set U, D, L, R, where U stands for "Up", etc. Tamassia finds the angles using network flows (in fact, he solves a more general problem of minimizing the number of bends in the drawing). At the heart of this method is the result that, given an angle assignment that is locally consistent (i.e., the angles at every vertex sum to $360°$), the graph has a rectilinear planar drawing if and only if each face cycle does [27]. As in the other cases we have discussed, the proof shows the stronger result that embedding choices for the faces can be combined arbitrarily. A cycle with an angle assignment has a rectilinear planar drawing if and only if the number of right turns minus the number of left turns is 4 in a clockwise traversal inside the cycle [27, 28].
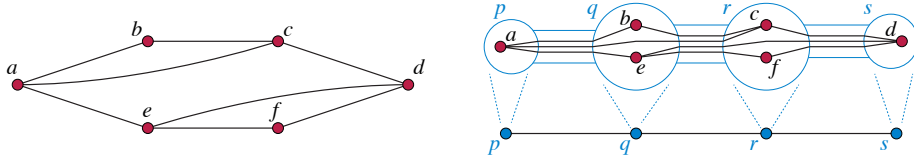
6

Fig. 6. A flat-folding of a six-vertex graph G (left), described as a self-touching configuration in which G is mapped onto a four-vertex path H (right), shown with magnified views of its edges and vertices of H.

Our result on flat folding can be used to prove an extension of the above result to rectilinear graph drawings with angles specified and with lengths assigned to the "horizontal" edges. (Note that the angle information allows us to distinguish the two classes of edges, although it is arbitrary which class is horizontal and which is vertical.) Given a rectilinear plane graph, contract all vertical edges, assigning angles of $0, 180°, 360°$ in the obvious way. Finally, in order to avoid "nested" vertices at the same coordinate as in Figure 5, we use the same trick of adding an extra edge incident to each vertex that has only incoming or only outgoing edges. We claim that the resulting multi-graph has a flat-folding if and only if the original has a rectilinear planar drawing with horizontal edges of the specified lengths. For the non-trivial direction, suppose we have a flat folding of the constructed graph. We must expand each vertex to a vertical line segment with the horizontal edges touching the line segment in a way that is consistent with the original graph. This can easily be done in a left to right sweep.

From this reduction we obtain the result that, if G is a plane graph of maximum degree 4 with specified angles that sum to $360°$ at each vertex and with lengths assigned to the horizontal edges, then G has a rectilinear planar drawing realizing these angles and edge lengths if and only if every cycle has a rectilinear planar drawing realizing the angles and lengths. Furthermore, we can combine any embedding choices for the faces that involve different vertical visibilities—although of course we cannot allow a vertical edge to have different lengths in its two cycles. This is a new result, to the best of our knowledge.

2   Definition

Following previous works in this area [10, 23] we formalize the notion of a flat folding using *self-touching configurations*. Intuitively, these are planar embeddings in which edges and vertices are allowed to be infinitesimally close to each other. A one-dimensional self-touching configuration of a graph G consists of a mapping from G to a path graph H that maps vertices of G to vertices of H and edges of G to paths in H, together with a *magnified view* of each vertex and edge of H that describes the local connectivity of the image of G at that point. In a self-touching configuration, the *multiplicity* of an edge in H is a positive integer, the number of different edges of G that map to it. The magnified view of an edge gives a linear ordering of the edges in its preimage. The magnified view

7

of a vertex v of H is a planar embedding of a neighborhood of the preimage of v into a disk, consistent with the magnified views of the edges incident to v.

By replacing each vertex of H by its magnified view, and each edge of H by a corridor of finite width through which each edge passes, it is possible to transform a self-touching configuration into a conventional planar drawing (with edges that may curve or bend) of the given graph G. We call this the *expanded drawing* of a self-touching configuration (Figure 6).

With this definition in hand, we may define a *flat folding* to be a self-touching configuration in which all edges of H lie on a single line. We consider two flat foldings to be equivalent if they have the same magnified views in the same order or in the reversed order as each other. A *face* of a flat folding or self-touching configuration is a cycle formed by a face of the expanded drawing. The *angle* formed by a pair of incident edges in a flat folding is one of three values, 0, 180°, or 360°, accordingly as the face lies between the two edges, the two edges extend in opposite directions from their common endpoint, or the two edges extend in the same direction with the face on both sides of them. An *angle assignment* to a plane graph is an assignment of the values 0, 180°, or 360° to each of its angles, regardless of whether this assignment is compatible with a flat-folding of the graph. An angle assignment is *consistent* if the angles sum to 360° at every vertex.

We define a *touching pair* of edges in a self-touching configuration of a graph G to be two edges e and f of G such that these two edges are consecutive in the magnified view of at least one edge in H. Each touching pair can be assigned to a single face of G, the face that lies between the two edges.

3 Local Characterization

In this section we show that for a plane graph with assigned lengths and consistent angles, being able to fold the whole graph flat is equivalent to being able to fold each of its faces flat.

Theorem 1. *Let G be a plane graph with given edge lengths and a consistent angle assignment. Then G has a flat folding if and only if every face cycle of G (with the induced assignment of lengths and angles) has a flat folding. More strongly, for every combination of flat foldings of the faces of G, there exists a flat folding of G itself whose touching pairs for each face are exactly the ones given in the folding of that face.*

Proof. In one direction, the proof is straightforward: if G has a flat folding, then this folding can be restricted to each face of G, and the set of touching pairs for the face is the same for the flat folding of this face as it is for the flat folding of the whole graph. For the remainder of the proof, we assume in the other direction that we have been given a set of flat foldings of the faces of G, and we use this assumption to show that G itself is flat foldable with the same touching pairs.

As described in Section 1.2, the assignment of lengths and angles given with G (together with an arbitrary choice of an x coordinate for one vertex and an

8

orientation for one edge) gives us a unique assignment of x coordinates for the vertices of G in any possible flat folding. We will start by subdividing all the edges of G. Take the set of x-coordinates of vertices of G and add an extra "half" x-coordinate at the midpoint between any two consecutive coordinate values. Subdivide each edge of G by adding vertices at all the x-coordinates in this set. The same subdivisions can be made in any flat folding of G, so there is no change to the existence or number of flat foldings. The subdivision does change the set of touching pairs, but two edges of the original graph form a touching pair if and only if two of the edges in the paths they are subdivided into form a touching pair, so the subdivision also does not change the correctness of the part of the theorem about touching pairs.

With G subdivided in this way, we carry out the proof by induction on the number of face angles that are assigned to be $360°$ (*mountain folds*). The base case of the induction is the case in which G has only two such angles, on the outer face; in this case it is an st-plane graph, each face has a unique flat folding (that coincides with its unique upward planar drawing), and the upward planar drawing of G gives a flat folding with the same touching pairs.

If G contains a vertex v, and an interior face f in which v is a mountain fold, then let e be one of the two edges of f incident to v, the one that is uppermost in the magnified view of the flat folding edge corresponding to these two edges, and let e' be the edge immediately above that one. (Edge e' must exist, because if e were the topmost edge in this magnified view, then f would necessarily be the exterior face.) Let v' be the endpoint of e' whose x-coordinate is the same as v. We form a graph G' by identifying v with v', ordering the edges of the combined super-vertex so that e' and e remain consecutive. (This produces a graph, not a multigraph, because the other endpoints of e and e' are subdivision points at a "half" x-coordinate, and so cannot coincide with each other.) This vertex identification reduces the number of mountain folds by one compared with G, and splits f into two simpler faces f_1 and f_2. The same split operation can be done to the flat folding of f, giving flat foldings of f_1 and f_2. Thus, G' meets the conditions of the lemma and has fewer mountain folds; by induction it has a flat folding realizing all the touching pairs of its face foldings, which are the same as the touching pairs of the face foldings of G. In this flat folding, the supervertex of G' formed from v and v' can be split back into the two separate vertices v and v', giving the desired flat folding of G.

The case when there exist three or more mountain folds on the exterior face is similar, but we must be more careful in our choice of v. Each mountain folded vertex on the exterior face is either a local minimum or local maximum of x-coordinates; because there are three or more of them, we may choose v to be a vertex that is not a unique global extremum. Then, as above, we find a vertex v' with the same coordinate, above or below v, and merge v and v' into a single vertex, giving a graph G' with fewer mountain folds in which the outer face has been split into two faces, one outer and one inner. As before, these two faces inherit a flat folded state from the given flat folding of the outer face of G, so by

induction G' has a flat folding. And as before, v and v' may be split back into separate vertices in this flat folding, giving the desired flat folding of G. $\square$

4 Algorithm to Find a Folding

For completeness, we briefly describe a greedy "crimping" strategy for finding flat-folded states of simple cycles with pre-assigned fold angles. Bern and Hayes [6] used a similar strategy to flat-fold cycles without pre-assigned angles. Arkin et al. [4] applied this method to open polygonal chains with assigned angles. The version here for cycles with assigned angles is described by Demaine and O'Rourke [12]. We do not describe its (non-trivial) correctness proof.

First, remove any flat folds from the input by merging the edges on either side of the fold. Then, repeatedly find an edge e such that the two edges on either side of e are at least as long as e, with folds of opposite type at its ends. If no such edge e exists, the cycle has no folding. If an edge e that meets these conditions can be found, it is safe to perform both folds, merging e with its two neighboring edges into a single edge of a simpler polygon.

Maintaining a set of edges that are ready to be folded, and performing each fold, takes constant time per fold, so folding a cycle in this way, and recovering the covering relation of its ordered line embedding, may be done in linear time. Putting the characterization from Section 3 together with the algorithm for folding a single cycle described above gives us an algorithm for testing whether a given plane graph G with edge length and angle assignment is flat foldable:

Theorem 2. *We can test flat foldability of a plane graph with given edge lengths and given angle assignment in linear time.*

Proof. We partition the graph into its component faces, and apply the crimping algorithm to an Euler tour of each face. Each face takes time proportional to its size, so the total time for the algorithm is linear. For the correctness of forming simple cycles from each face by taking Euler tours, see Appendix A. $\square$

5 Counting Flat Foldings

We cannot use crimping to count the flat foldings of a cycle, because some flat foldings cannot be formed by a sequence of crimping steps (Figure 7). Instead, to count flat foldings in a single cycle, we use dynamic programming.

Lemma 1. *Given a single n-vertex cycle, with an assignment of edge lengths and angles, it is possible to count the flat foldings of the cycle in polynomial time.*

For the proof, see Appendix B.

Theorem 3. *We can count the flat foldings of an n-vertex planar graph G with an assignment of edge lengths and angles in polynomial time.*
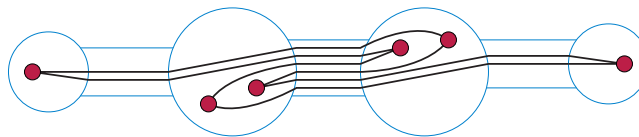
10

Fig. 7. Magnified view of a flat folding that cannot be obtained by crimping.

Proof. We apply Lemma 1 to the Euler tour of each face of G and return the product of the resulting numbers. □

References

1. T. G. Abbott, E. D. Demaine, and B. Gassend. arXiv:0901.1322, January 2009, http://arxiv.org/abs/0901.1322.
2. Z. Abel, E. D. Demaine, M. L. Demaine, S. Eisenstat, J. Lynch, T. B. Schardl, and I. Shapiro-Ellowitz. Folding equilateral plane graphs. *Internat. J. Comput. Geom. Appl.* 23(2):75–92, 2013, doi:10.1142/S0218195913600017.
3. H. Alt, C. Knauer, G. Rote, and S. Whitesides. On the complexity of the linkage reconfiguration problem. *Towards a Theory of Geometric Graphs*, pp. 1–13. Amer. Math. Soc., Contemp. Math. 342, 2004, doi:10.1090/conm/342/06126.
4. E. M. Arkin, M. A. Bender, E. D. Demaine, M. L. Demaine, J. S. B. Mitchell, S. Sethia, and S. S. Skiena. When can you fold a map? *Comput. Geom. Th. Appl.* 29(1):23–46, 2004, doi:10.1016/j.comgeo.2004.03.012.
5. B. Ballinger, D. Charlton, E. D. Demaine, M. tin L. Demaine, J. Iacono, C.-H. Liu, and S.-H. Poon. Minimal locked trees. *Proceedings of the 11th Algorithms and Data Structures Symposium*, pp. 61–73, Lecture Notes in Computer Science 5664, August 2009.
6. M. Bern and B. Hayes. The complexity of flat origami. *Proc. 7th ACM-SIAM Symposium on Discrete algorithms (SODA '96)*, pp. 175–183, 1996.
7. P. Bertolazzi, G. Di Battista, G. Liotta, and C. Mannino. Upward drawings of triconnected digraphs. *Algorithmica* 12(6):476–497, 1994, doi:10.1007/BF01188716.
8. T. Biedl, E. D. Demaine, M. L. Demaine, S. Lazard, A. Lubiw, J. O'Rourke, S. Robbins, I. Streinu, G. Toussaint, and S. Whitesides. A note on reconfiguring tree linkages: trees can lock. *Discrete Appl. Math.* 117(1-3):293–297, 2002, doi:10.1016/S0166-218X(01)00229-3.
9. S. Cabello, E. D. Demaine, and G. Rote. Planar embeddings of graphs with specified edge lengths. *J. Graph Algorithms & Appl.* 11(1):259–276, 2007, doi:10.7155/jgaa.00145.

10. R. Connelly, E. D. Demaine, and G. Rote. Infinitesimally locked self-touching linkages with applications to locked trees. *Physical Knots: Knotting, Linking, and Folding Geometric Objects in* $\mathbb{R}^3$ *(Las Vegas, NV, 2001)*, pp. 287–311. Amer. Math. Soc., Contemp. Math. 304, 2002, doi:10.1090/conm/304/05200.

11. R. Connelly, E. D. Demaine, and G. Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete & Computational Geometry* 30(2):205–239, September 2003.

12. E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra.* Cambridge University Press, 2007, doi:10.1017/CBO9780511735172.

13. G. Di Battista and E. Nardelli. Hierarchies and planarity theory. *IEEE Trans. Systems Man Cybernet.* 18(6):1035–1046, 1988, doi:10.1109/21.23105.

14. C. A. Duncan and M. T. Goodrich. Planar orthogonal and polyline drawing algorithms. *Handbook of Graph Drawing and Visualization*, chapter 7, pp. 223–246. Chapman and Hall/CRC, 2013.

15. A. Estrella-Balderrama, J. J. Fowler, and S. G. Kobourov. Characterization of unlabeled level planar trees. *Comput. Geom. Th. Appl.* 42(6-7):704–721, 2009, doi:10.1016/j.comgeo.2008.12.006.

16. A. Garg. New results on drawing angle graphs. *Comput. Geom. Th. Appl.* 9(1):43–82, 1998, doi:10.1016/S0925-7721(97)00016-3.

17. A. Garg and R. Tamassia. Upward planarity testing. *Order* 12(2):109–133, 1995, doi:10.1007/BF01108622.

18. A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.* 31(2):601–625, 2001, doi:10.1137/S0097539794277123.

19. M. Harrigan and P. Healy. Practical level planarity testing and layout with embedding constraints. *Proc. 15th Int. Symp. Graph Drawing (GD 2007)*, pp. 62–68. Springer, Lecture Notes in Comput. Sci. 4875, 2008, doi:10.1007/978-3-540-77537-9_9.

20. T. C. Hull. The combinatorics of flat folds: a survey. *Origami*^{3} *(Asilomar, CA, 2001)*, pp. 29–38. A K Peters, 2002, arXiv:1307.1065.

21. M. Jünger, S. Leipert, and P. Mutzel. Level planarity testing in linear time. *Proc. 6th Int. Symp. Graph Drawing (GD '98)*, pp. 224–237. Springer, Lecture Notes in Comput. Sci. 1547, 1998, doi:10.1007/3-540-37623-2_17.

22. J. Pach and G. Tóth. Monotone drawings of planar graphs. *J. Graph Theory* 46(1):39–47, 2004, doi:10.1002/jgt.10168, arXiv:1404.5892.

23. A. Ribó Mor. *Realization and counting problems for planar structures.* Ph.D. thesis, Free Univ. Berlin, 2006.

24. J. Saxe. Embeddability of weighted graphs in k-space is strongly NP-hard. *Proc. 17th Allerton Conf. Commun. Control Comput.*, pp. 480–489, 1979.

25. M. Schaefer. Realizability of graphs and linkages. *Thirty Essays on Geometric Graph Theory*, pp. 461–482. Springer New York, 2013, doi:10.1007/978-1-4614-0110-0_24.

26. I. Streinu and W. Whiteley. Single-vertex origami and spherical expansive motions. *Revised Selected Papers from the Japan Conference on Discrete and Computational Geometry*, pp. 161–173, Lecture Notes in Computer Science 3742, October 2004, http://dx.doi.org/10.1007/11589440_17.

27. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.* 16(3):421–444, 1987, doi:10.1137/0216030.

28. G. Vijayan and A. Wigderson. Rectilinear graphs and their embeddings. *SIAM J. Comput.* 14(2):355–372, 1985, doi:10.1137/0214027.
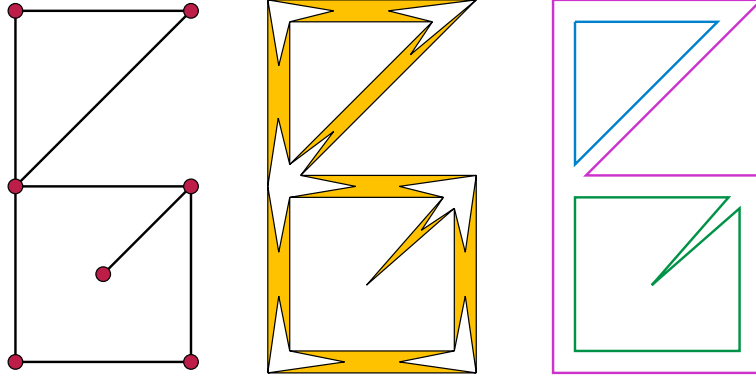
Fig. 8. Left: a graph G that is not 2-vertex-connected, and has repeated vertices and edges on its three faces. Center: replacing each dangling edge by a 4-cycle, and each other edge by a 6-cycle, produces a 2-vertex-connected graph H with an equivalent set of flat foldings. Right: The Euler tours of the faces of G are the simple cycles forming the corresponding faces of H.

A Euler Tours for Nonsimple Faces

Our algorithms for finding and counting flat foldings of graphs rely on subroutines for solving the same problem on simple cycles, the faces of the given graph. However, if a plane graph is not 2-vertex-connected, its faces may not actually be simple cycles: if an articulation vertex or bridge edge appears on the boundary of a face, it may appear multiple times on that face. As an extreme instance of this phenomenon, we may have a tree as our input graph; in this case, it has a single face, with every edge repeated twice and every vertex repeated a number of times equal to its degree.

In this case, we may obtain a collection of simple cycles that represent the faces of the graph by an *Euler tour* technique, as follows. Make two copies of each edge of the given embedded graph G, one for each of the two sides of the edge (regardless of whether these two sides are both part of a single face or whether they belong to two different faces). Then, form closed walks by connecting each of these doubled edges to the edges clockwise and counterclockwise of the edge within the same face. Replace any repeated vertex in these walks by new vertices, so that each face of G gives rise to a simple cycle. In this cycle, copy the edge lengths and angles from G.

Lemma 2. *Each face of the given graph G is flat-foldable if and only if each of the simple cycles resulting from the Euler tour technique described above is flat-foldable. The number of flat-foldings of G is equal to the product, over these simple cycles, of the number of different patterns of internal visibilty that can be formed from a flat-folding of the cycle.*

13

Proof. Given the graph G, we form a new graph H from G, by replacing each edge of G by a cycle, as shown in Figure 8. In more detail:

– A dangling edge e, having one endpoint of degree one, becomes a concave four-cycle, with one vertex at the position of that degree-one endpoint adjacent to two vertices at the other endpoint, and with one concave vertex inside the triangle formed by these other three edges. We assign the two edges connecting to the degree-one endpoint the same lengths as the original length of e, and the other two edges are assigned a shorter length ε. Within this four-cycle, the angles are assigned as a valley fold (0) at the three convex vertices and a mountain fold (360°) at the concave vertex.

– An edge e that is not dangling is replaced by a non-convex hexagon; two opposite edges of this hexagon, of length equal to e, connect a pair of new vertices at each endpoint of e, and each of these pairs is connected by a path of two length-ε edges through a new vertex between the two long edges of the hexagon. Within the face formed by this new hexagon, we assign the angles as a valley fold at the four convex vertices and a mountain fold at the concave vertex.

– At each vertex v of G of degree $d > 1$, identify pairs of vertices from the edge polygons to form a face of $2d$ vertices, alternating between vertices placed at the original location of v and vertices that (in the folded state) should be ε away from that location. Label the angles of this face with valley folds at the vertices ε distance away from v. Each remaining vertex of this face is formed by identifying a pair of vertices from the polygons of two edges e and f. Let θ be the angle formed at v by edges e and f, and within the face for v label the corresponding angle by $360° - \theta$. This system of labels ensures that, at all vertices of the expanded graph, the angle labels correctly add to $360°$.

Each new 4-cycle or 6-cycle formed in this way is flat-foldable in a unique way, so the expansion of the edges of G into cycles does not affect the flat-foldability or the number of flat foldings of the resulting graph. Each face of the resulting graph is either an Euler tour of a face of G, or one of the new 4-cycles or 6-cycles. As the 4-cycles and 6-cycles are uniquely flat-foldable, we do not need to test or count flat-foldings of these faces. The result follows by applying Theorem 1 to the expanded graph. □

B Proof of Lemma 1

We prove here the claim that the number of flat foldings of the interior of a simple cycle (with assigned lengths and angles) may be counted by dynamic programming in polynomial time.

We use the given assignment to compute x-coordinates for all vertices, and subdivide the cycle as in the proof of Theorem 1. After this step, no mountain folded vertex is interior to the range of x-coordinates of another edge, and each input edge is subdivided into a path of at least two edges. By the argument used

14

to prove Theorem 1, a flat-folding of the cycle is equivalent to set of identifications between pairs of vertices within the cycle, such that these identifications partition the cycle into smaller polygons each of which has exactly two valley folds and zero mountain folds.

To count sets of identifications of this type, we use a dynamic programming algorithm that counts, for each ordered pair u, v of vertices with the same x-coordinate as each other, the numbers $A(u, v)$ and $B(u, v)$ of flat-foldings (if any) of the polygon formed by identifying u with v and keeping the part of the input from u clockwise to v. Here we only consider pairs such that that the edge e immediately clockwise from u and the edge f immediately counterclockwise from v both extend in the same direction, left or right, and such that the identification of u with v produces a valley fold rather than a mountain fold. $A(u, v)$ will count the total number of flat-foldings of this polygon, while $B(u, v)$ will count only a subset of the flat-foldings, the ones in which edges e and f are visible to each other rather than being blocked by a mountain-fold vertex with the same x-coordinate as both u and v. By symmetry we can assume without loss of generality that edges e and f both lie to the left of u and v, so in any flat-folded state u and e must be below v and f. See Figure 9.
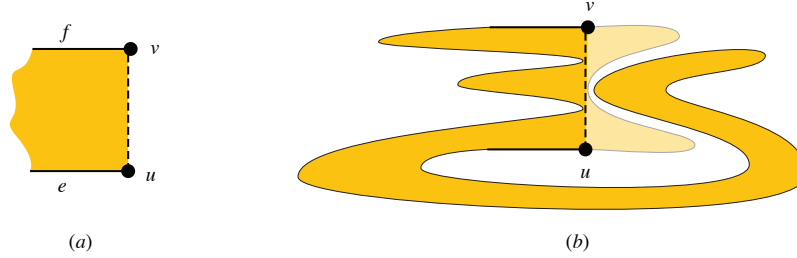

Fig. 9. (a) The dynamic programming subproblem for u and v. (b) A possible solution to the subproblem, and its place in the whole solution (light colour).

To compute $A(u, v)$, we sum $B(u, v)$ together with the number of folds in which a mountain vertex w blocks e from f. We will check all possible choices of w; to avoid double-counting, we will only consider folds where w is the topmost such mountain vertex (the one closest to v). Thus, for each choice of w, we include in our sum the term $A(u, w)B(w, v)$ counting the number of folds in the two sub-polygons between u and w and between w and v. That is, we may calculate $A(u, v)$ by the formula

$$A(u, v) = B(u, v) + \sum_{w} A(u, w)B(w, v)$$

where the sum is over the possible choices of mountain vertices that are between u and v, have their two adjacent edges to the left, and have the same x-coordinate as u and v. See Figure 10(a).

To compute $B(u, v)$, we consider the two vertices u' and v' at the other ends of the incident edges e and f. Because of the way we subdivided the input, u' and v' again have the same x-coordinate as each other. If $u' = v'$, we are done and there is exactly one flat-folded state: $B(u, v) = 1$. If either u' or v' is a valley fold, there can be no valid flat-folded state of the polygon between u and v, and again we are done: $B(u, v) = 0$. And if u' and v' are both flat vertices, then we can copy the value from a simpler subproblem: $B(u, v) = A(u', v')$.

In the remaining cases, one or both of u' and v' is a mountain fold. Consider the case in which u' is a mountain fold and v' is not; the other cases are similar. See Figure 10(b). Let g be the other edge incident to u'. Because u' is a mountain fold, edge g lies to the right of u'. In any flat-folded state, u' must see another non-mountain vertex w below it, from the sub-polygon between u and v (ignoring intervening mountains). Identifying u' and w splits the polygon from u to v into two parts, one to the right of the identified vertex and one to the left. The right sub-polygon can be described by a subproblem from u' to w, and the left sub-polygon can be described by a subproblem from w to v'. Thus, in this case, we have the formula

$$B(u, v) = \sum_w A(u', w) A(w, v')$$

where again the sum is over all suitable choices of w. In the case where both u' and v' are mountain folds, we have a similar sum over two choices of intermediate vertex, one below u' and one above v', forming two right-pointing subproblems with one left-pointing subproblem between them.

There are a polynomial number of subproblems, each involves a summation with a polynomial number of choices, and the arithmetic operations in these summations involve numbers with a linear number of bits. Therefore, the total time is polynomial.   □
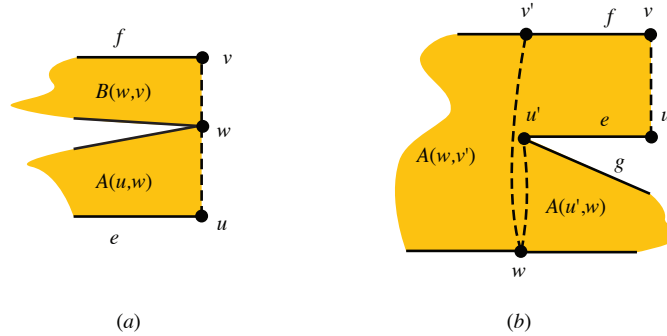

(a)   (b)

Fig. 10. (a) Computing $A(u, v)$ by trying all possible choices for w. (b) Computing $B(u, v)$ in case u' is a mountain but v' is not.