Making Polygons by Simple Folds and One Straight Cut

Erik D. Demaine* Martin L. Demaine* Andrea Hawksley* Hiro Ito†
Po-Ru Loh* Shelly Manber* Omari Stephens*

Abstract. We give an efficient algorithmic characterization of simple polygons whose edges can be aligned onto a common line, with nothing else on that line, by a sequence of all-layers simple folds. In particular, such alignments enable the cutting out of the polygon and its complement with one complete straight cut. We also show that these makable polygons include all convex polygons possessing a line of symmetry.

1 Introduction. Take a sheet of paper, fold it flat, and then make one complete straight cut. What shapes can the unfolded pieces have? This *fold-and-cut problem* was introduced formally at JCDCG'98 [3], motivated by a 1922 magic trick by Harry Houdini, but with history going back to a 1721 Japanese puzzle book. The answer is that any pattern of straight-line-segment cuts can be obtained in this way [3, 2, 4, ch. 17].

We consider a special case of the fold-and-cut problem, called *simple fold-and-cut*, where we require the folding process to consist of a sequence of all-layers simple folds. Given an existing flat folded state, an *all-layers simple fold* [1]. consists of folding along a line, through all layers of paper met by the line, by $\pm 180°$, so that afterward all paper is planar and on one side of the line. We call a graph *simple-fold-and-cuttable* if a sequence of all-layers simple folds brings the graph's vertices and edges to a line, with no excess paper on that line.

We prove the two main theorems.

Theorem 1 *There is a strongly polynomial-time algorithm for determining whether a given (not necessarily convex) simple polygon is simple-fold-and-cuttable, starting from a convex piece a paper strictly containing the polygon.*

The polynomial running time is a function of the number n of vertices in the input polygon, even though the number of required simple folds can be arbitrary large for a fixed n; see Fig. 1. As a result, when a polygon is simple-fold-and-cuttable, the algorithm produces only an implicit representation of the folding sequence. An explicit representation can be obtained, at the cost of adding to the running time a term linear in the output size.

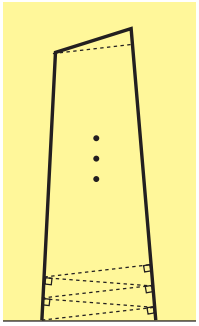

Figure 1: As the two acute base angles approach $\pi/2$, the number of simple folds grows without bound.

Theorem 2 *A convex polygon is simple-fold-and-cuttable if and only if it has a line of reflectional symmetry.*

2 Definitions. The input to the (simple-)fold-and-cut problem consists of a *piece of paper* P, which we require to be a polygonal region,[1] and a graph G drawn on P with edges drawn as straight line segments. We view each vertex v of G as a point of P, and thus each edge $\{v, w\}$ of G corresponds to the segment vw, which we require to be contained in P. In fact, we can also view G as a subset of P, namely the set of all points on vertices and edges of G. From this perspective, we naturally remove any degree-2 vertices forming an angle of $180°$, as we can simply merge to two incident edges. For convenience, we refer to such a drawing (G, P) as a *graffito*.[2]

We make two assumptions about the input graffito (G, P). First, we require that P is convex; typically, we have in mind that P is a square or a rectangle. Second, we assume that G does not touch the boundary of P; we say that (G, P) has a *margin*. Both conditions are required only of the initial graffito, as they can easily become violated after making one or more folds; we will take care to preserve the first property, but the second property will remain violated after the first fold.

For a graffito (G, P), we call an all-layers simple fold *feasible* if no point of G folds on top of a point in $P \setminus G$; in other words, every point of G folds to either another point of G or a point outside P. Because we consider all-layers simple folds, we can effectively glue together multiple layers of paper into one layer, resulting in a new polygonal region P' of paper, as well as a new graph G' drawn on P'. We write $(G, P) \to (G', P')$ when a feasible all-layers simple fold takes (G, P) to (G', P'). We write $(G, P) \to^* (G', P')$ to denote zero or more transitions $(G, P) \to \cdots \to (G', P')$, allowing in particular $(G, P) = (G', P')$.

A graffito (G, P) is *cuttable* if there is a straight line ℓ such that $G \subset \ell$ (all vertices and edges of G are on ℓ) and $P \setminus G$ is disjoint from ℓ (the rest of the paper is off ℓ). We call a graffito (G, P) *simple-fold-and-cuttable* if $(G, P) \to^* (G', P')$ for some cuttable graffito (G', P').

Our model of computation is a real RAM.

3 Algorithm. How do we determine simple-fold-and-cuttability of a graffito (G, P) where G is a simple polygon, and P is convex and has a margin around G? The first step is to find G's lines of reflectional symmetry, because the first feasible fold must be such a line: a simple fold reflects one side onto the other, and cannot map G to anything other than G because of the margin. All such symmetry lines can be found in $O(n)$ time, where n is the number of vertices in the polygon [5]. In particular, there are at most $O(n)$ lines of symmetry. Our algorithm tries each of these lines as the first fold, and for each spends $O(n^2)$ time testing whether the remainder is simple-fold-and-cuttable, for a total time of $O(n^3)$.

*MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, {edemaine,mdemaine,hawksley,ploh}@mit.edu, shellym@alum.mit.edu, xsdg@xsdg.org

†School of Informatics, Kyoto University, Kyoto 606-8501, Japan, itohiro@i.kyoto-u.ac.jp

[1] We treat a piece of paper as a closed region, namely, the set of all points interior or on the boundary of the polygon.

[2] "Graffito" is the singular form of "graffiti" (both in English and Italian), though it seems rarely used in English.

Suppose the first fold is $(G, P) \rightarrow (G', P')$. Then (G', P') is a *passage*, meaning that it is a graffito satisfying the following three conditions: G is a simple polygonal line, the two endpoints of G are on the boundary of P, and no point of G except the two endpoints is on the boundary of P. The last property follows from our assumption that P has a margin around G.

It remains to characterize simple-fold-and-cuttable passages in $O(n^2)$ time. Our algorithm follows a greedy approach, repeatedly making feasible folds, until either getting stuck or the result is cuttable. This idea is motivated by the following lemma, which states that feasible folds can never hurt.

Lemma 1 *If (G, P) is a passage, P is convex and has a margin around G, and $(G, P) \rightarrow (G', P')$, then (G, P) is simple-fold-and-cuttable if and only if (G', P') is simple-fold-and-cuttable.*

Proof sketch: Because of the passage property, one side of the fold has a shorter part of G, and because of the margin property, this part folds on top of the larger part and effectively disappears; thus $G' \subset G$. (This is the critical point where we use that the original graph is a polygon.) But we may not have $P' \subset P$. To fix this, we fold P' down to a convex subset of $P \cap P'$. Now any simple-fold sequence for (G, P) applies as well to (G', P'). □

A high-level description of our passage algorithm is as follows. Initially we mark the endpoints of G as "real"; in the future, the endpoints may become marked as "limit" (meaning that the end edges are in fact slightly longer, but can be shortened arbitrarily close to reaching this endpoint). Throughout, we maintain the invariant that (G, P) is a passage. The algorithm repeatedly loops through the following steps. First, we replace P with the convex hull of G, with the understanding that every boundary point of this hull should in fact be surrounded by a small (infinitesimal) neighborhood of paper, except for the real endpoints of G which are truly on the paper boundary. (At limit endpoints, the end edges of G extend in the same direction infinitesimally beyond the limit endpoints, to reach the boundary of P.) Now we look for feasible folds that either hit a vertex of G other than an endpoint, or cross an edge of G other than an end edge. If there is at least one such fold, we arbitrarily chose one and fold it, marking the new endpoint as real. If there are no such folds but there are feasible folds through end edges, we compute the limit of repeatedly folding folds that cross just the end edges (as detailed below), and mark any modified endpoints as limits. If there are no feasible folds whatsoever, then (G, P) is simple-fold-and-cuttable if and only if it is cuttable, i.e., a single edge. In the former two cases, we proceed to the next iteration of the loop, trimming to the convex hull and searching for the next suitable fold.

This description leaves out a few algorithmic details:

3.1 Computing the convex hull.. We can compute the convex hull of the polygonal line G in linear time using, e.g., Melkman's algorithm. The intuition behind replacing P with the convex hull of G (plus an infinitesimal "fringe") is that we can always fold P down to be arbitrarily close to this hull without touching G, and this may create new feasible folds.

3.2 Finding feasible folds that hit a non-end vertex/edge.. For each non-end vertex and the midpoint of each non-end edge, we test foldability of the angular bisector (perpendicular for midpoints) in $O(n)$ time by walking in both directions from the starting point, testing for symmetry locally around each pair of vertices visited (taking into consideration local neighborhoods of boundary points), until passing an endpoint in either direction, after which feasibility is guaranteed by the convexity of P. The total time is $O(n^2)$.

3.3 Computing the limit of folds that cross just end edges.. We distinguish two cases according to whether the two rays extending the end edges beyond G's endpoints intersect.

First consider the case in which the rays do not intersect. For each end edge $e = v_0v_1$ with end vertex v_0, we project the union of the non-end edges (which form a path starting at v_1) onto the line through e, obtaining in $O(n)$ time a line segment s containing v_1. No fold through e that crosses only end edges can pass through s; thus, if s contains e, then no fold across e is possible. Otherwise, the endpoint of s closer to v_0 is a strict upper bound on the extent to which e can be folded.

If the above procedure provides information that some or all of an end edge e' cannot be folded, then we project the nonfoldable segment of e' onto the opposite end edge e and update the bound for e accordingly. (If we have nonfoldability information about both end edges, then we apply the above step twice, using each edge to update the other.) We show that the updated bounds are the desired limits, that is, the end edges can be folded to within infinitesimal amounts of their updated bounds, and in particular, updating the bound for e can give us no new bound on the foldability of e'.

The situation is slightly different if the rays extending the end edges do intersect at a point X: in this case, it is possible that the above approach would cause an infinite cascade of back-and-forth bound updates on the two end edges. Fortunately, in this case, the end edge farther from X is actually not foldable at all, so it suffices to apply the projection method once to determine the foldability of the end edge closer to X.

4 Convex Polygons. Proof sketch of Theorem 2: The "only if" part is trivial. After the first simple fold, we are left with a passage (G, P) where G is a convex polygonal line. At least one of the end edges of G can be made arbitrarily small by folding through it. Once an end edge is smaller than its adjacent edge, we can fold the end edge onto its adjacent edge, reducing the number of edges. Repeat. □

References.

[1] 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? *Computational Geometry: Theory and Applications*, 29(1):23–46, 2004.

[2] M. Bern, E. Demaine, D. Eppstein, and B. Hayes. A disk-packing algorithm for an origami magic trick. In *Origami^3: Proceedings of the 3rd International Meeting of Origami Science, Math, and Education*, pp. 17–28, 2001. Also FUN 1998.

[3] E. D. Demaine, M. L. Demaine, and A. Lubiw. Folding and cutting paper. In *Revised Papers from the Japan Conference on Discrete and Computational Geometry*, pp. 104–117, 1998.

[4] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge Univ. Press, 2007.

[5] J. D. Wolter, T. C. Woo, and R. A. Volz. Optimal algorithms for symmetry detection in two and three dimensions. *The Visual Computer*, 1(1):37–48, 1985.