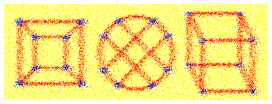
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 24, no. 2, pp. 89–103 (2020)
DOI: 10.7155/jgaa.00520

# Efficient Algorithm for Box Folding

*Koichi Mizunashi*[1] *Takashi Horiyama*[1] *Ryuhei Uehara*[2]

[1]Saitama University, Japan
[2]Japan Advanced Institute of Science and Technology, Japan

**Abstract**

For a given polygon $P$ and a polyhedron $Q$, the folding problem asks if $Q$ can be obtained from $P$ by folding it. This simple problem is quite complicated, and there is no known efficient algorithm that solves this problem in general. In this paper, we focus on the case that $Q$ is a box, and the size of $Q$ is not given. That is, input of the box folding problem is a polygon $P$, and it asks if $P$ can fold to boxes of certain sizes. We note that there exist an infinite number of polygons $P$ that can fold into three boxes of different sizes. In this paper, we give a pseudo polynomial time algorithm that computes all possible ways of folding of $P$ to boxes.

| Submitted: | Reviewed: | Revised: | Accepted: |
|---|---|---|---|
| April 2019 | July 2019 | August 2019 | October 2019 |
| | Final: | Published: | |
| | January 2020 | February 2020 | |
| Article type: | | Communicated by: | |
| Regular paper | | K. Mukhopadhyaya and S.-i. Nakano | |

A preliminary version of this paper was presented to the 13th International Conference and Workshops on Algorithms and Computation (WALCOM 2019), Guwahati, India, 2019. This work is partially supported by MEXT/JSPS KAKENHI Grant Numbers 17H06287, 18H04091, 18K11153, and the Asahi Glass Foundation.

*E-mail addresses:* mizunashi@al.ics.saitama-u.ac.jp (Koichi Mizunashi) horiyama@al.ics.saitama-u.ac.jp (Takashi Horiyama) uehara@jaist.ac.jp (Ryuhei Uehara)

# 1 Introduction

In 1525, the German painter Albrecht Dürer published his masterwork on geometry [7], whose title translates as, "On Teaching Measurement with a Compass and Straightedge for lines, planes, and whole bodies." In the book, he presented each polyhedron by drawing a *net* for it: an unfolding of the surface to a planar layout. To this day, it remains an important open problem whether every convex polyhedron has a (non-overlapping) net by cut along edges. When we allow to cut not only along edges, this problem is settled only for tetramonohedron, which is a kind of tetrahedron: any net of a tetramonohedron is characterized by a (non-overlapping) tiling [3].

To understand unfolding, it is interesting to look at the inverse: one folding problem asks what polyhedra can be folded from a given polygonal sheet of paper. For example, the Latin cross, which is one of eleven nets of a cube, can form 23 different convex polyhedra (including doubly covered convex polygons) by 85 distinct ways of folding (and an infinite number of doubly covered concave polygons). Comprehensive surveys of folding and unfolding can be found in [6]. Recently, Abel et al. investigated the folding problem of bumpy pyramids [1]: For a given petal polygon $P$ (convex $n$-gon $B$ with $n$ triangular petals), it asks if we can fold to a pyramid (with flat base $B$) or a convex bumpy pyramid by folding along a certain triangulation of $B$. In [1], they gave nontrivial linear time algorithms for the problem.

Let us elaborate on this and its related results. Alexandrov's Theorem states that every metric with the global topology, and local geometry required of a convex polyhedron is, in fact, the intrinsic metric of some convex polyhedron. Thus, if $P$ is a net of a convex polyhedron, the shape is uniquely determined. Alexandrov's Theorem was stated in 1942, and a constructive proof was given by Bobenko and Izmestiev in 2008 [4]. A pseudo-polynomial time algorithm for Alexandrov's Theorem was given by Kane, Price, and Demaine in 2009 [9]. However, it runs in $O(n^{456.5}r^{1891}/\epsilon^{121})$ time, where $r$ is the ratio of the largest and smallest distances between vertices, and $\epsilon$ is the coordinate relative accuracy. The exponents in the time bound of the result are remarkably huge. As far as the authors know, the results for the bumpy pyramids are the first efficient algorithms for Alexandrov's Theorem for a family of nontrivial convex polyhedra.

In this paper, for a given polygon $P$, we consider the box folding problem that asks if $P$ can fold to a box or not. This problem seems to be natural and simple from the viewpoint of our life. We first note that this problem is motivated by counterintuitive polygons. In 1999, Biedl et al. found two polygons that can fold into two different boxes [6, Figure 25.53]. Later, Mitani and Uehara proved that there exist an infinite number of polygons that can fold into two boxes [10], and Shirakawa and Uehara proved that there exist an infinite number of polygons that can fold into three boxes [11]. So far, a polygon that can fold into three different boxes in four different ways has been found by using a supercomputer (Figure 1), and it is open that there exists a polygon that can fold into $k$ different boxes for $k > 3$ [12].

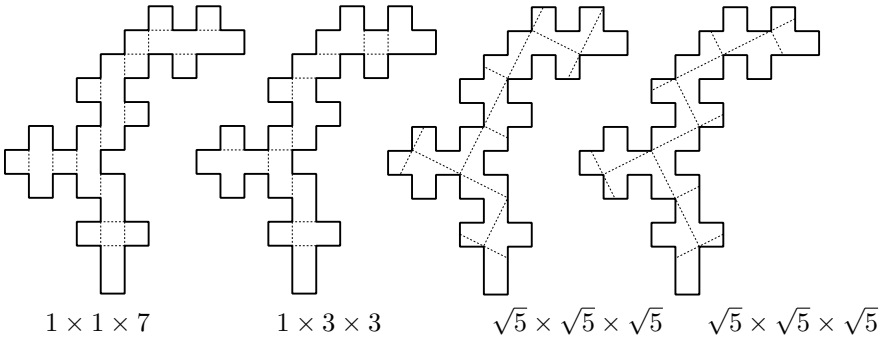


Figure 1: A polygon that can fold into three different boxes of sizes $1 \times 1 \times 7$, $1 \times 3 \times 3$, and $\sqrt{5} \times \sqrt{5} \times \sqrt{5}$ in four different ways [12].

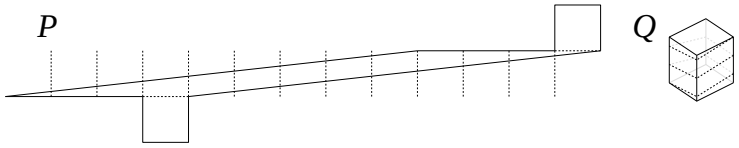


Figure 2: A geometric parameter $m$ that the number of line segments on an edge of a cube $Q$. In this example, an edge of $Q$ consists of $m = 4$ line segments, however, it can be arbitrarily large if the slope of $P$ is more slanted.

In the previous research, they did not solve the box folding problem in general form. The results in [6, Figure 25.53] and [11] were obtained without a computer. In [10], they assume that a polygon $P$ is a polyomino, which is a union of unit squares sharing on their edges, and they only search the boxes obtained by folding along the edges of unit squares. In [12], they first obtain the set of all polyominoes of area 30 that can fold into two boxes of sizes $1 \times 1 \times 7$ and $1 \times 3 \times 3$ on the similar model in [10]. Then they solved the box folding problem for the special box of size $\sqrt{5} \times \sqrt{5} \times \sqrt{5}$ on these polyominoes. Namely, it is specialized to this special size (see [12] for the details).

Recently, Horiyama and Mizunashi solved the box folding problem in more general case with supporting parameters [8]; the input polygon $P$ is a polyomino, and the size $a \times b \times c$ of the box $Q$ is also given. Moreover, the matching of edges (that gives us the gluing of the corresponding edges) of $P$ is also given. In this case, the box folding problem can be solved in $O((n+m)\log n)$ time, where $m$ is the maximum number of line segments on an edge of the folded box $Q$. We here note that this geometric parameter $m$ is independent from the number of vertices in $P$ and $Q$. Even in a simple example in Figure 2, $m$ can be arbitrarily large while $P$ and $Q$ have 10 and 8 vertices, respectively.

In this paper, from both viewpoints of theoretical and practical, we give an efficient algorithm for the box folding problem in general. That is, the input is a polygon $P$ with $n$ vertices. Then the output is the set of whole boxes $Q$ folded

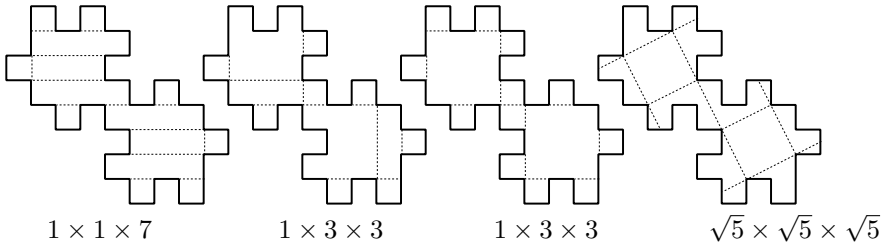


Figure 3: Another polygon that can fold into three different boxes of sizes $1 \times 1 \times 7$, $1 \times 3 \times 3$, and $\sqrt{5} \times \sqrt{5} \times \sqrt{5}$ in four different ways not mentioned in [12].

from $P$ with distinct ways of folding. The algorithm runs in pseudo-polynomial time of $n$ and $D$, where geometric parameter $D$ is the diameter of $P$.

We show a case study of our algorithm for the nine polyominoes shown in [12]. In [12], the authors first compute the set of polyominoes that can fold two boxes of sizes $1 \times 1 \times 7$ and $1 \times 3 \times 3$. There are 1080 polyominoes. Then they solved the box folding problem for the special box of size $\sqrt{5} \times \sqrt{5} \times \sqrt{5}$ on these 1080 polyominoes. Finally, they found nine polyominoes that can fold into three different boxes. We apply our algorithm to this set of nine polyominoes as a case study. Surprisingly, among them, we have another special polyomino that can fold into three different boxes in four different ways of folding (Figure 3). We will discuss the reason why the authors of [12] missed finding it, while we succeed.

## 2 Preliminaries

In this section, we first state the box folding problem:

**Input :** A polygon $P = (p_0, p_1, \ldots, p_{n-1}, p_0)$
**Output:** A set $S = \{Q_0, Q_1, \ldots, Q_k\}$ of boxes that can be folded from $P$

Note that $S$ can be an empty set. Let $x(p_i)$ and $y(p_i)$ be the $x$-coordinate and $y$-coordinate of a point $p_i$, respectively. We assume that $x(p_i)$ and $y(p_i)$ are rational numbers for each $i = 0, \ldots, n-1$. Then we have the following observation:

**Observation 1** *Assume that each $x(p_i)$ and $y(p_i)$ in $P$ are described by rational numbers. Let $q_{\min}$ be the least common denominator of them. Scaling up by $q_{\min}$, the box folding problem can be solved for $P' = (p'_0, p'_1, \ldots, p'_{n-1}, p'_0)$ such that each $x(p'_i)$ and $y(p'_i)$ in $P'$ are integers in $[0, p_{\max} q_{\min}]$, where $p_{\max}$ is the largest numerator.*

Therefore, hereafter, we assume that each coordinates $x(p_i)$ and $y(p_i)$ are non-negative integers without loss of generality. For the polygon $P$, its *diameter*

$D$ is defined by $\max_{i,j} |p_i - p_j| = \max_{i,j} \sqrt{(x(p_i) - x(p_j))^2 + (y(p_i) - y(p_j))^2}$. Here we introduce another geometric parameter $m$ that indicates the number of line segments on an edge of $Q$. This is independent of the number of vertices in $P$ and $Q$. In a simple example in Figure 2, $m$ can be arbitrarily large.

Now, we turn to the definition of a box and its development and net. Let $Q$ be a convex polyhedron. It is a *box* if $Q$ consists of three pairs of rectangular faces. It results that $Q$ has 6 faces, 8 vertices, and 12 edges. At each vertex, its *curvature* is 270°, and at any other point, its *curvature* is 360°[1]. When we cut $Q$ along a set of polygonal lines, unfold on a plane, and obtain a polygon $P$, the set is called a *development* of $Q$. We assume that any cut ends at a point with curvature less than 360°. Otherwise, it makes a redundant cut on $P$, which can be reduced. The development $P$ is called a *net* of $Q$ if and only if $P$ is a connected simple polygon without self-overlap or hole. Let $T$ be the set of cut lines on $Q$ to obtain a net $P$. Then the following is well known (see [6, Sec. 22.1.3] for details):

**Theorem 1** *$T$ forms a spanning tree of the vertices of $Q$.*

In this paper, the following theorem is useful:

**Theorem 2** *Let $Q$ be a box and $P$ be a net of $Q$. Then (1) all vertices of $Q$ appear on the boundary of $P$, and (2) $P$ has at least two vertices of degree 270°, which correspond to two vertices of $Q$.*

**Proof:** By Theorem 1, the set of cut lines on $Q$ forms a spanning tree $T$. We note that each edge of $T$ appears twice on the boundary of $P$. If a vertex of $Q$ appears inside of $P$, $P$ cannot be flat. Therefore, we obtain (1) immediately. When the vertex $v$ of $Q$ has $\deg(v) \geq 2$ on $T$, the vertex will be cut into $\deg(v)$ pieces and spread on the boundary of $P$. In this case, $v$ appears $\deg(v)$ times, and $P$ has less than 270° at these points. Let $\ell$ be a leaf of $T$. Then $\ell$ corresponds to a vertex of $Q$; otherwise, the curvature around $\ell$ is 360°, and it does not make a boundary of $P$. Thus around at $\ell$, the curvature is 270°. Since $T$ has at least two leaves, we have (2). □

As shown in [12, Theorem 2], for any positive integer $k$, there are a series of boxes $Q_i$ of size $a_i \times b_i \times c_i$ for $i = 1, 2, \ldots, k$ such that all distinct $Q_i$ have the same area. However, once an area is given, we have an upper bound of the number of such boxes:

**Observation 2** *Let $P$ be a polygon of area $A$, and it can fold into some boxes. Then the number of possible edge lengths of boxes is $O(A^{2/\log\log A})$.*

**Proof:** Let $a, b, c$ be the edge lengths of a box $Q$ of area $A$ with $a \leq b \leq c$. Then we have $2(ab + bc + ca) = A$. Since $A$ is an integer by Observation 1, each of $a, b, c$ can be represented by $i\sqrt{j}$ for some positive integers $i$ and $j$. By $a \leq b \leq c$

[1] We do not give the formal definition of curvature. Intuitively, it indicates the total angle of paper surrounding a vertex by opening the vertex out in our context. See [6, Sec. 21.2] for further details.

and $2(ab+bc+ca) = A$, we have $6a^2 \leq A$, which implies the number of possible $a$ is $O(A)$ since $A$ is an integer, and $a$ is $i\sqrt{j}$ for some positive integers $i$ and $j$ in general. In fact, the number of divisors of $A$ is $O(A^{1/\log\log A})$. Then the number of possible $b$ is also $O(A^{1/\log\log A})$ since $2b^2 \leq 2(ab+bc+ca) = A$. Once $a$ and $b$ are fixed, $c$ is uniquely determined. Therefore, the number of possible edge lengths $(a, b, c)$ is $O(A^{2/\log\log A})$. □

# 3 Algorithm Description

In this section, we describe our algorithm. We first give an outline of the algorithm, and show the details about proof techniques.

## 3.1 Outline of Algorithm

The outline of our algorithm is simple:

```
Input  : A polygon P = (p_0, p_1, ..., p_{n-1}, p_0)
Output: A set S = {Q_0, Q_1, ..., Q_k} of boxes that can be folded from
        P
foreach box Q of size a × b × c such that 2(ab + bc + ca) is equal to area
 of P do
    for i ← 0 to n − 1 do
        if curvature at p_i is 270° then
            Check if P is a net of Q such that p_i is a vertex of Q;
        end
    end
end
```

That is, the algorithm checks all possible points $p_i$ if it makes 270°. By Theorem 2, if $Q$ can be folded from $P$, there are at least two points that fold to the vertices of $Q$. Hereafter, we assume that $p_0$ is the vertex of $Q$ without loss of generality. This time, for the given $P$ and $p_0$, the algorithm checks if $P$ can be folded into the box $Q$. This step is a loop for direction of $Q$ as follows:

1. First, fix the direction of $Q$ on $P$ in an arbitrary way. That is, the algorithm first arbitrarily chooses a direction of $Q$ on $P$ at the vertex $p_0$.

2. Then the algorithm checks if all vertices of $Q$ are on the boundary of $P$. This is a necessary condition.

3. If any vertex $v$ of $Q$ is inside of $P$, the algorithm rotates the direction of $Q$ clockwise at the center $p_0$ to move $v$ to the boundary of $P$. Repeat this rotation until no vertex of $Q$ is inside of $P$.

4. Check if each vertex of $Q$ makes 270° in total.

5. Finally, the algorithm checks these vertices can be glued to fold the box $Q$. If they can be glued to the box $Q$, output it.

6. The algorithm rotates the direction of $Q$ clockwise to find the next position. If the rotation makes 360°, the algorithm halts.

Intuitively, the algorithm checks all possible positions of $P$ to fold into $Q$. There are two major points to be considered. The first point is how to check if $P$ can fold to $Q$ at the position and direction. In order to check this point, we will use a technique named "stamping". The second point is the number of rotations of $Q$. We will show that the number of rotations can be bounded above by a polynomial of $n$ and some geometric parameters.

Hereafter, we assume that $p_0$ of $P$ coincides with a vertex of $Q$, and it makes an angle 270°. Let $v_0 = p_0$ be a vertex of $Q$ and $v_1, v_2, v_3$ be three vertices of $Q$ adjacent to $v_0$ on $Q$ by three edges (or crease lines) $v_0v_1$, $v_0v_2$, and $v_0v_3$ in clockwise. Without loss of generality, we assume that $|v_0v_1| = a$. Then we have two cases that either $|v_0v_2| = b$ and $|v_0v_3| = c$ or $|v_0v_3| = b$ and $|v_0v_2| = c$. The algorithm will check these two cases. Here we suppose that $|v_0v_2| = b$ and $|v_0v_3| = c$ since the other case is symmetry. We describe the details of above two points and show the analysis of the algorithm.

## 3.2 Stamping

The algorithm first adjusts $p_0$ of $P$ on $v_0$ of $Q$ with the assumption the line $v_0v_3$ is cut. From this position, it rotates the relative position of $Q$ centered at $v_0 = p_0$ to search a proper direction that satisfies the necessary conditions for the vertices of $P$ and $Q$. We will show that the number of these rotations can be bounded by a polynomial of some geometric parameters. Let denote the angle of rotation by $\theta = \angle v_3p_0p_1$ at $p_0 = v_0$. That is, the algorithm starts from $\theta = 0°$ and updates $\theta$. When $\theta \geq 360°$, it finishes to check.

For a given angle $\theta$, the algorithm has to check whether (1) all vertices of $Q$ are on the boundary of $P$, and (2) for each vertex $v_i$ of $Q$, the curvatures corresponding to the points on the boundary of $P$ sum up to 270°. In order to do that, the algorithm has to follow the corresponding points to the vertices of $Q$ on $P$.

The basic idea is called *stamping* in [2]. In [2], Akiyama rolls a regular tetrahedron on a plane as a *stamper* and obtains a tiling by the stamping. The key property of the stamping in [2] is that a regular tetrahedron has the same direction and position when it returns to the original position, no matter what the route was. Therefore, the cut lines of any net on the surface of a regular tetrahedron tile plane, or the net tiles plane.

We use a box $Q$ as a stamper on $P$. In this case, when the stamper $Q$ returns to the original position, its face and direction change according to the route. In fact, it is used for designing puzzles, and its complexity is investigated [5]. However, the key properties we use here are that a box $Q$ is orthogonal, and each coordinate is an integer, which are useful to bound the number of possible cases. As used in [2], when we roll $Q$ on an edge $e$ of $Q$ on $P$, this operation corresponds to develop $Q$ to $P$. That is, when we draw the cut lines of $Q$ on $Q$ and stamp it on the plane, it corresponds to develop $Q$ into the net $P$.

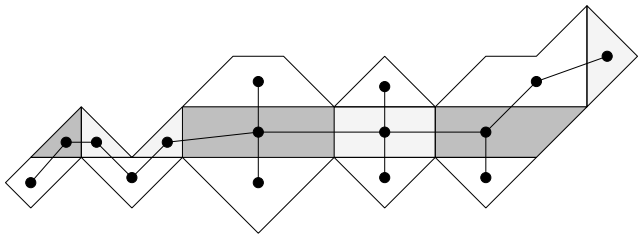

Figure 4: Tree structure of $P$: Each face of the box $Q$ is cut into "particles". Then the adjacent relationship of the particles induces a tree on $P$.

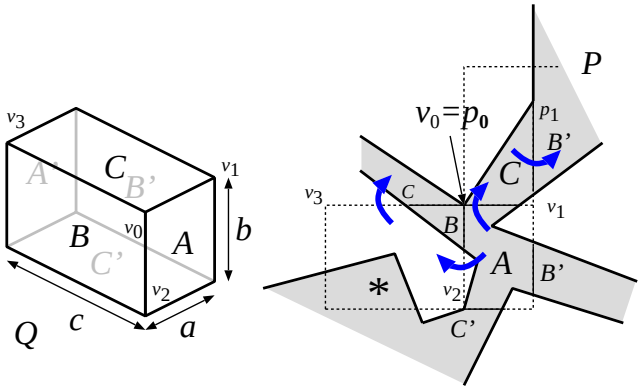


Figure 5: Rolling a box $Q$ on $P$. When we roll it left and up, we have $B$ and $C$. When we roll it up and right, we have $C$ and $B'$.

We will use the tree structure of $P$ defined as follows (Figure 4). Each face of the box $Q$ is cut into "particles" when it is developed to $P$. In other words, $P$ is partitioned into particles by the edges of $Q$ (or folding lines of $P$). On $P$, the particles correspond to the vertices, and two vertices are joined by an edge if and only if the corresponding particles share an edge of positive length on $Q$. Then since $P$ is a simple polygon and all vertices of $Q$ are on the boundary of $P$, the resulting graph induces a tree. Essentially, the algorithm performs the breadth first search on this tree by rolling the box $Q$ on $P$, and it obtains the partition of $P$ into the particles by stamping of $Q$.

A simple example is given in Figure 5. The stamper $Q$ has six different labels $A$, $A'$, $B$, $B'$, $C$, and $C'$. (They are just labels to distinguish with each other, and we do not mind the direction when it is "stamped" on $P$.) The stamper $Q$ starts at the initial position: It is on the face $X$ that contains $v_0 = p_0$, and the corner (90°) is covered by $P$ without cut. In the case of Figure 5, the initial position is either on the polygons labeled $A$ or $B$ since the face $C$ of $Q$ is cut into two pieces on $P$.

We assume that we have the curvature 270° at $v_0 = p_0$. Therefore, we always

have at least two candidates for the initial position (when $\theta = 0$, we have three candidates). We choose any one as the initial position. In this case, assume that we choose the face $A$ with $v_0 = p_0$ as an initial position and put $Q$ at the initial position.

Let denote the intersection of $P$ and the face of $Q$ on $P$ by $Q \cap P$. Then, at the initial position, $Q \cap P$ gives us the area ($\subset P$) labeled $A$. When $Q$ is rolled up and right, $Q \cap P$ gives us the areas labeled $C$ and $B'$, respectively. On the other hand, when $Q$ is rolled (from the initial position) left and up, $P \cap Q$ gives us the areas labeled $B$ and $C$, respectively.

Here, this stamping (or labeling ) should be *continuous*. Precisely, when $Q$ is rolled on an edge $e$, the resulting polygons in $Q \cap P$ obtain their labels only if they share the edge $e$ with the labeled area before rolling. (In the context of the tree structure shown in Figure 4, the labeling is done for a polygon in $Q \cap P$ only when it is adjacent to the labeled neighbor.) In the case of Figure 5, when $Q$ is rolled left from the initial position, the area labeled by $B$ obtains its label since it shares an edge with the area labeled by $A$. On the other hand, the area with label $*$ does not obtain any label this time.

Intuitively, the stamping sweeps over $P$ from the initial position along $P$. We repeat rolling from the initial position until all points in $P$ are included in the labeled polygons. When all points in $P$ are in the labeled polygons[2], we say $Q$ *stamps* $P$. We say that the stamping is *feasible* if no vertex $v_i$ of $Q$ is put inside of $P$ in the stamping. We will consider if $P$ is a net of $Q$ for the current $\theta$. In this situation, $P$ may be a net of $Q$ only if $P$ is feasible by Theorem 2.

**Lemma 1** *Assume that $Q$ and $\theta$ give us a feasible stamping of $P$. We also assume that $P$ is a net of $Q$. Then, each point in $P$ obtains a unique label (except on the edges of $Q$). That is, the stamping gives us a partition of $P$ along the edges of $Q$.*

**Proof:** To derive a contradiction, we assume that $p \in P$ obtains two labels. Then there are two different sequences $S_1$ and $S_2$ of rolling of $Q$ to stamp on $p$. Without loss of generality, $S_1$ and $S_2$ share up to the same face $F_0$, and then $Q$ is rolled up on the face $F_1$ in $S_1$ and $Q$ is rolled right on the face $F_2$ in $S_2$. Let $v$ be the top right corner of $F_0$, which is shared by $F_1$ and $F_2$. If $v$ is inside of $P$, the stamping is not feasible. Therefore, $v$ is on the boundary of $P$. Now we consider the point $q$ at $(x(v) + \epsilon, y(v) + \epsilon)$ for $0 < \epsilon < 1$. Since $P$ is a simple polygon, $q$ is outside of $P$.

In this case, the sequences $S_1$ and $S_2$ will share the point $p$ such that $q$ is outside of $P$ (Figure 6). Then the union of rectangles in $S_1$ and $S_2$ from $F_0$ to $p$ surrounds the initial position. Therefore, $P$ should contain a hole, or a point shared by two or more rectangles. The former case contradicts that $P$ is a net of $Q$, and the latter case contradicts that $P$ is feasible. Therefore, the stamping gives us a partition of $P$. □

[2]For sake of simplicity, we do not define the labels of points in $P$ on an edge shared by two rectangles of $Q$. We also do not define the label of a point $p$ corresponding to the vertex $v_i$ of $Q$.

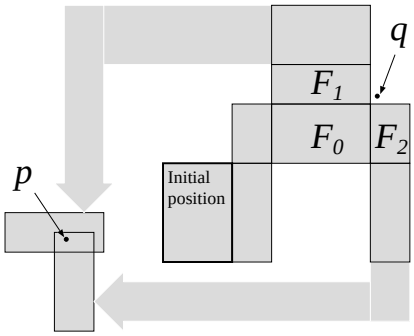


Figure 6: Stamping of $Q$ generates a partition of $P$.

**Lemma 2** *Let $D$ be the diameter of $P$. Then the total number of stampings of $Q$ on $P$ at the vertex $p_0 = v_0$ by the algorithm is $O((D/a)^6 n)$.*

**Proof:** We first note that the algorithm contains two different steps. First, the algorithm puts $Q$ at the initial position. If the initial position is feasible, it is okay. However, in general, we have to seek the first feasible position. After the first step, we have to seek the next feasible position from the current feasible position.

In the first step at the initial position, when the stamping is not feasible, the algorithm first finds a vertex of $Q$ inside of $P$. This is done from the initial position by stamping by the breadth first search manner. Let $v$ be the first vertex of $Q$ inside of $P$. Then the algorithm finds the first boundary of $P$ by rotating $\delta$ for the smallest rotation angle. Precisely, after rotation of angle $\delta$, $v$ is moved on the boundary of $P$. This can be done by computing the locus of $v$ as a part of the circle centered at $v_0$ with radius $|vp_0|$. After the rotation, the algorithm checks if it has a feasible stamping for the new angle. If it is not feasible by some vertex of $Q$ inside of $P$, the algorithm repeats the process until there are no such vertices of $Q$ inside of $P$.

In the second step, now all vertices of $Q$ are on the boundary of $P$ (or outside of $P$). By the stamping, the algorithm also knows the correspondence between the vertices of $Q$ and the vertices on the boundary of $P$. Therefore, the algorithm checks whether each vertex of $Q$ has curvature of $270°$ in total. If all vertices are of curvature $270°$, the algorithm performs the check of gluing in the next phase. After checking the gluing, the algorithm has to rotate for finding the next angle of stamping such that no vertex is inside of $P$ after the rotation of, say, $\delta'$. We assume that $\delta' > 0$ and the rotation direction is clockwise. For each vertex $v_i$ on the boundary of $P$, $v_i$ is rotated along the circle centered at $v_0$ with radius $|v_0 v_i|$. If the right side of $v_i$ on the circle is not inside of $P$, we do not take care of it. When it is inside, we define $v_i'$ by the rightmost vertex on the circle such that any other vertex $v$ between $v_i$ and $v_i'$ is inside of $P$. Intuitively, when $\delta'$ is less than the angle $\angle v_i v_0 v_i'$, this vertex $v_i$ will be inside of $P$, it is not a feasible position after rotation. Therefore, $\delta'$ is the maximum angle among

these vertices on the boundary of $P$. After rotation of angle $\delta'$, the algorithm performs the new stamping of $Q$ over $P$ and checks if it is feasible or not. If it is not, the algorithm performs the first step again. If all vertices are on the boundary of $P$, it performs this second step again.

Now we turn to consider the total number of the rotations. We here note that during the rotation, each vertex $v$ of $Q$ can be put on an edge $e$ of $P$ at most twice. (It can occur twice only when the circle centered at $p_0$ with radius $|vp_0|$ has two intersection points with the edge $e$.)

Therefore, the total number of the repeating of these steps can be bounded above by the summation of the number of intersection points on the circle centered at $p_0$ with radius $|vp_0|$ for each vertex $v$ of the stamping.

We note that the number of the vertices $v$ of a stamping depends on $P$ and $\theta$, and $m$. We here show an upper bound of this number. To consider this number, we switch the role of $P$ and $Q$. The algorithm rotates the direction of $Q$, but now we imagine that the direction of $Q$ is fixed, and the polygon $P$ is fixed. In this case, we roll $Q$ like a dice on $P$. Since $P$ is of diameter $D$, it is enough to roll the die $Q$ in a square of size $D \times D$. Then the set of vertices of $Q$ forms a subset all possible points in this square of size $D \times D$. Let $p$ be any of these possible points in this square. Then its coordinates can be explained as $x(p) = k_1a + k_2b + k_3c$ and $y(p) = h_1a + h_2b + h_3c$ with $0 \leq x(p) \leq D$ and $0 \leq y(p) \leq D$ for some nonnegative integers $k_1, k_2, k_3, h_1, h_2, h_3$. Then $k_1a+k_2b+k_3c \leq (k_1+k_2+k_3)a$, which implies $k_1+k_2+k_3 \leq D/a$. Therefore, the number of possible $x(p)$ can be bounded above $(D/a)^3$. By the same argument, the number of possible $y(p)$ is bounded above by $(D/a)^3$. Thus, the total number of possible points of $Q$ on $P$ is $O((D/a)^6)$. On the other hand, as discussed above, each of these $O((D/a)^6)$ points can be put on one of $n$ edges at most twice for each rotation. Therefore, the number of rotations in the algorithm is $O((D/a)^6 n)$ in total. □

We note that the upper bound $O((D/a)^6)$ of the number of rotations is pessimistic. For example, when $a = b = c$, it is reduced to $O((D/a)^2)$.

We also note that each feasible stamping gives us the whole vertices $v_i$ of $Q$ on the boundary of $P$. Therefore, we can check if each vertex $v_i$ has a curvature $270°$ in total in linear time of $n$. Therefore, after the first phase, we know that $P$ is partitioned into particles of faces of $Q$ with their corresponding labels, and each vertex $v_i$ has a curvature $270°$ in total.

### 3.3 Check of gluing

In this phase, we check if we fold to $Q$ by $P$ along the crease lines given in the first phase[3].

Hereafter, we sometimes consider the polygon $P = (p_0, p_1, \ldots, p_{n-1}, p_0)$ consists of vectors $\overrightarrow{p_0p_1}, \overrightarrow{p_1p_2}, \ldots, \overrightarrow{p_{n-1}p_0}$ for the sake of simplicity. Then we can deal with "gluing of two edges" by an operation of vectors. For example, in

[3]Some readers may consider the first phase is enough. However, we have not yet checked if some particles of polygons cause overlap on a face of $Q$. In other words, we have to check each face is made by particles of polygons by gluing without overlap or hole.

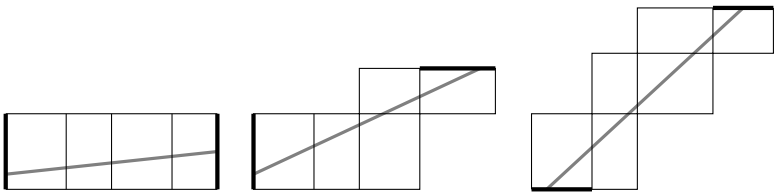

Figure 7: Examples of stamping $Q$ along an edge $e$ (gray lines) of $P$.

Figure 5, we assume that we glue two edges $p_0p_1$ and $p_0p_{n-1}$. Then, we have three cases after gluing:
(1) $|p_0p_1| > |p_0p_{n-1}|$: We obtain $\overrightarrow{p_{n-1}p_1}$ such that $|p_{n-1}p_1| = |p_0p_1| - |p_0p_{n-1}|$.
(2) $|p_0p_1| < |p_0p_{n-1}|$: We obtain $\overrightarrow{p_{n-1}p_1}$ such that $|p_{n-1}p_1| = |p_0p_{n-1}| - |p_0p_1|$.
(3) $|p_0p_1| = |p_0p_{n-1}|$: We obtain $p_{n-1} = p_1$.

We here remind that if $P$ is a net of $Q$, the set of line segments of cut on $Q$ forms a spanning tree $T$ (Theorem 1). Moreover, each edge of $T$ appears twice on the boundary of $P$. Now we know that $v_0 = p_0$ is the corner of $Q$ of size $a \times b \times c$, and which line is $v_0v_1$ of length $a$ by $\theta$. Therefore, from this point $v_0$, we can glue edge by edge and check if $Q$ can be folded from $v_0$ by $P$. The details of this part can be found in [8], and it can be done in $O((n+m)\log n)$ time, where $m$ is the maximum number of line segments in an edge of $Q$ (Each line segment corresponds to two adjacent particles of two faces of $Q$). Now, we give an upper bound on $m$.

**Lemma 3** *Let $m$ be the maximum number of line segments in an edge of $Q$ for all $Q \in \mathcal{Q}$, where $\mathcal{Q}$ is the set of boxes of the same surface area with $P$. Then, $m$ has an upper bound $m = O(Dn)$, where $D$ is the diameter of $P$, and $n$ is the number of vertices of $P$.*

**Proof:** Suppose that edge $L$ of some $Q$ gives $m$, i.e., $L$ is divided into $m$ line segments by the stamping. We can obtain an upper bound on $m$ by counting the number of crossings of $L$ and the edges of $P$. An edge $e$ of $P$ may go across several stamped copies of $L$. Figure 7 illustrates some examples of $e$ between two copies of $L$, where gray lines denote line segments of $e$, and bold lines denote the copies of $L$. The stamped faces are obtained by rolling $Q$ up or right along the line segments of $e$. In any case (including the cases not illustrated in Figure 7), a line segment of $e$ between two copies of $L$ is included in at least four stamped faces, and its length is at least 1 (the unit length). Since the length of $e$ is bounded by $D$, the number of line segments of $e$ divided by the copies of $L$ is $O(D)$, which means that $L$ and $e$ have $O(D)$ crossings. Therefore, as $P$ has $n$ edges, $L$ crosses the edges of $P$ at most $O(Dn)$ times. □

## 3.4 Analysis of algorithm

The correctness of the algorithm follows from the discussion above. Therefore, we show that the algorithm runs in pseudo-polynomial time.

**Theorem 3** *For a given polygon $P$ of $n$ vertices, the algorithm solves the box folding problem in $O(D^{11}n^2(D^5 + \log n))$ time, where $D$ is the diameter of $P$.*

**Proof:** We first consider the main loop. The number of combinations of the size $a \times b \times c$ of a box is given by $O(A^{2/\log\log A})$, where $A$ is the area of $P$. For each trio $(a, b, c)$ with $a \leq b \leq c$, the main loop checks for each point $p_i$ of $P$ of angle 270°. There are $O(n)$ such angles in the worst case.

For each $p_i = v_0$, the algorithm performs the stamping of $Q$ on $P$. It is not difficult to see that it can be done in $O(M)$ time, where $M$ is the number of rectangles tiled over $P$ to cover it.

Now we turn to the time complexity of the checking of vertices. First, the algorithm puts a box $Q$ on $P$ so that $v_0 = p_0$ and $\theta = 0$. Then it finds the next feasible stamping if it is not.

The first step is that while there is a vertex of $Q$ inside of $P$, repeat rotations until all vertices of $Q$ are on $P$. Finding the first vertex $v$ inside of $P$ takes $O(Mn)$ time; the stamping by the breadth first search takes $O(M)$ rollings and checks if a vertex of the rectangle face is in $P$ takes $O(n)$ time.

Once it finds a position of $Q$ on $P$ such that no vertex is inside of $P$, the algorithm computes $\delta'$, which is the maximum angle that needs to move $v_i$ not inside of $P$ for each $v_i$. For each vertex $v_i$, the computation of the corresponding $v_i'$ on the circle centered at $v_0$ of radius $|v_0v_i|$ takes $O(n)$ time (along the edges of $P$). The number of vertices of $v_i$ on the boundary of $P$ is $O(M)$. Therefore, this step takes $O(Mn)$ time.

Once we find a feasible stamping, we have to check if $P$ can be glued to $Q$. This step takes $O((n+m)\log n)$ time, where $m$ is the maximum number of line segments on an edge of the folded box $Q$ by using the algorithm in [8].

Thus, the running time of this algorithm is $O(A^{2/\log\log A}(Mn)(Mn + (n + m)\log n))$ time for each phase. By Lemma 2, the total summation of $M$ is $O((D/a)^6)$. Using $A^{2/\log\log A} = O(D^4)$, it is simply that $O(D^4(D/a)^6((D/a)^6 n + (n+m)\log n)n)$ time. Now taking $a = 1$ and $m = O(Dn)$, we have the theorem. □

In Theorem 3, we show that our algorithm runs in $O(D^{11}n^2(D^5 + \log n))$ time. It is much efficient than the pseudo-polynomial time algorithm given by Kane, Price, and Demaine in 2009 [9], which runs in $O(n^{456.5}r^{1891}/\epsilon^{121})$ time. In addition to that, from the practical viewpoint, our algorithm runs efficiently. In the next section, we show the case study about polyominoes of area 30. We examined several polyominoes of area 30. In these experiments, $44 \leq n \leq 58$, and our program runs in less than one second for each case.

# 4 Case Study

The authors investigated the case that $P$ is an orthogonal polygon. We can assume that $P$ is a polyomino made of unit squares by refining, which simplifies the implementation of the algorithm (the details are omitted). In [12], Xu et al. found nine polyominoes of area 30 that can fold into three boxes of size

$1 \times 1 \times 7$, $1 \times 3 \times 3$, and $\sqrt{5} \times \sqrt{5} \times \sqrt{5}$. In our case study, $n < 60$ and each computation takes less than one second. In [12], the authors said that "Interestingly, one of nine such polygons folds into three different boxes $1 \times 1 \times 7$, $1 \times 3 \times 3$, and $\sqrt{5} \times \sqrt{5} \times \sqrt{5}$ in four different ways." The polygon with four different ways of folding is shown in Figure 1.

However, their claim is not correct. There is another polyomino that has the same property as shown in Figure 3. That is, the precise claim is as follows. Among polyomino of area $30$[4], there are nine polyominoes that can fold into three different boxes of these sizes. Among these nine, two polyominoes have four different ways of folding into three different boxes, and seven polyominoes have three (unique) different ways of folding into three different boxes.

The reason why the authors of [12] missed finding one is hidden in their algorithm. Their first algorithm found all polyominoes that folded into two boxes of sizes $1 \times 1 \times 7$ and $1 \times 3 \times 3$. There are 1080 polyominoes that fold into these two boxes. This time, they did not consider how many ways of folding into two boxes. (Or they never thought that there might have been a polyomino that could fold into two boxes in three or more different ways.) For these 1080 polyominoes, their second algorithm checked all ways of folding and found nine polyominoes that fold into the third box of size $\sqrt{5} \times \sqrt{5} \times \sqrt{5}$. Since their second algorithm produced all ways of folding, as serendipity, they found that there was a polyomino that folded into the box of size $\sqrt{5} \times \sqrt{5} \times \sqrt{5}$ in two different ways. This is why they concluded that only one polyomino had 4 different ways.

[4]The number of polyomino of area 30 is 2,368,347,037,571,252.

## References


[1] Z. R. Abel, E. D. Demaine, M. L. Demaine, H. Ito, J. Snoeyink, and R. Uehara. Bumpy pyramid folding. *Computational Geometry*, 75:22 – 31, 2018. doi:10.1016/j.comgeo.2018.06.007.

[2] J. Akiyama. Tile-makers and semi-tile-makers. *The American Mathematical Monthly*, 114(7):602–609, 2007. doi:10.1080/00029890.2007.11920450.

[3] J. Akiyama and C. Nara. Developments of Polyhedra Using Oblique Coordinates. *J. Indonesia. Math. Soc.* , 13(1):99–114, 2007. doi:10.22342/jims.13.1.77.99-114.

[4] A. I. Bobenko and I. Izmestiev. Alexandrov's theorem, weighted Delaunay triangulations, and mixed volumes. *Annales de l'Institut Fourier*, 58(2):447–505, 2008. doi:10.5802/aif.2358.

[5] K. Buchin, M. Buchin, E. D. Demaine, M. L. Demaine, D. El-Khechen, S. Fekete, C. Knauer, A. Schulz, and P. Taslakian. On Rolling Cube Puzzles. In *19th Canadian Conference on Computational Geometry (CCCG 2007)*, pages 141–144, 2007.

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

[7] A. Dürer. *Underweysung der messung, mit den zirckel un richtscheyt, in Linien ebnen unnd gantzen corporen*. 1525. URL: http://books.google.com/books?id=5fxOAAAAcAAJ.

[8] T. Horiyama and K. Mizunashi. Folding Orthogonal Polygons into Rectangular Boxes. In *In Proc. of 19th Japan-Korea Joint Workshop on Algorithms and Computation (WAAC 2016)*, 2016.

[9] D. Kane, G. N. Price, and E. D. Demaine. A pseudopolynomial algorithm for Alexandrov's Theorem. In *11th Algorithms and Data Structures Symposium (WADS 2009)*, pages 435–446. Lecture Notes in Computer Science Vol. 5664, Springer-Verlag, 2009. doi:10.1007/978-3-642-03367-4_38.

[10] J. Mitani and R. Uehara. Polygons Folding to Plural Incongruent Orthogonal Boxes. In *Canadian Conference on Computational Geometry (CCCG 2008)*, pages 39–42, 2008.

[11] T. Shirakawa and R. Uehara. Common Developments of Three Incongruent Orthogonal Boxes. *International Journal of Computational Geometry and Applications*, 23(1):65–71, 2013. doi:10.1142/S0218195913500040.

[12] D. Xu, T. Horiyama, T. Shirakawa, and R. Uehara. Common Developments of Three Incongruent Boxes of Area 30. *Computational Geometry: Theory and Applications*, 64:1–17, 2017. doi:10.1016/j.comgeo.2017.03.001.