# Geodesics, Shortest Paths, and Non-Isomorphic Nets

Potvin Nicolas Zolotov Boris

ULB, December 2020

## 1 Introduction

Given a collection of 2D polygons, a *gluing* or a *net* describes a closed surface by specifying how to glue (a part of) each edge of these polygons onto (a part of) another edge. Alexandrov's uniqueness theorem [1] states that any valid gluing that is homeomorphic to a sphere and that does not yield a total facial angle greater than $2\pi$ at any point, corresponds to the surface of a unique convex 3D polyhedron (doubly covered convex polygons are also regarded as polyhedra). Note that the original polygonal pieces might need to be folded to obtain this 3D surface.

Unfortunately, the proof of Alexandrov's theorem is highly non-constructive. The only known approximation algorithm to find the vertices of this polyhedron [9] has a (pseudopolynomial) running time really large in $n$, where $n$ is the total complexity of the gluing.

There is no known exact algorithm for reconstructing the 3D polyhedron, and in fact the coordinates of the vertices of the polyhedron might not even be expressible as a closed formula [8].

Enumerating all possible valid gluings is also not an easy task, as the number of gluings can be exponential even for a single polygon [5]. However one valid gluing can be found in polynomial time using dynamic programming [7, 10]. Complete enumerations of gluings and the resulting polyhedra are only known for very specific cases such as the Latin cross [6] and a single regular convex polygon [7].

The special case when the polygons to be glued together are all identical regular $k$-gons, and the gluing is *edge-to-edge* was studied recently for $k \geq 6$ [3] and $k = 5$ [2]. The aim of this project is to study the case of $k = 4$: namely, to *enumerate* all valid gluings of squares and *classify* them up to isomorphism.

## 2 Methods

### 2.1 Interpreting and listing valid gluings

Due to Alexandrov's theorem, for every gluing of squares satisfying Alexandrov's conditions there is a convex polyhedron corresponding to it. It is possible to draw each of its faces on square grid, the vertices of the face being also the vertices of the grid.

Due to Gauss—Bonnet formula, the number of vertices of the polyhedron is at most 8,

which yields that the number of faces is at most 12. When there is a constant number of faces and edges, any kind of check can be carried for them in a short matter of time.

Square grid gives a convenient way to represent gluing as a set of faces draw on the grid, it is easy to check if Alexandrov's conditions are satisfied for a gluing if it is represented as a set of planar faces on the grid. This representation allows for estimating the number of valid gluings and for development of an algortihm with polynomial running time that lists all the gluings, which will be done later in this paper.

We have to note however, that a gluing can comply with different nets, and with one net in several different ways. See Figure 11 for an example: the net consisting of two parallelograms clearly corresponds to a doubly covered rectangle $1 \times 4$. This is still our a-priori knowlegde. It is not possible to determine which net is exactly the net of the polyhedron corresponding to the gluing, because of non-constructiveness of Alexandrov's theorem.

## 2.2 Chen—Han algorithm

Chen—Han algorithm is presented in [4]. Considering the hull $H$ of a convex polyhedron (*i.e.* its surface), the goal is to find the shortest path from one point of $H$ to another. To achieve it, the algorithm makes the simple observation that the shortest path follows a geodesic, hence the only angles are those made by moving from one face to another. In other words, if the polyhedron is cut open and flattened in such a way that the shortest path is not cut in the process, one would see this path as a straight line.

Following this observation, the idea of the algorithm is to project a cone of all possible paths from the source into all neighbour faces. Then from the projection of the source on an edge to the opposite edges of the face and so on. The sequences of these projections form corridors delimited by two geodesics through which goes the shortest paths from the source to any point on the surface.

This algorithm runs in $O(n^2)$ and uses up to $\Theta(n)$ space [4]. This algorithm can be used not only for the case when the polyhedron is cut into its faces, but into arbitrary flat parts. For the latter case, however, additional observations have to be made to show that the running time is preserved. For the case of a polyhedron that is glued of squares edge-to-edge this is done in [11].

### 2.2.1 Concepts and notions

Several key concepts are used in the Chen & Han algorithm. Each face of a polyhedron is modelled as a list of three oriented edges. These edges are sorted such that the start of an edge is the end of the previous one and conversely.

A *net* is a convenient way to represent the faces of a polyhedron. In essence, it simply is a list of $n$ triples of edges. That is a list of the $n$ faces modelled by their three edges.

Sorting the edges of a face is easy since they are all coplanar. In a three-dimensional object, one willing to organise the faces needs a notion different than order (in a mathematical sense). To tackle this difficulty, the edges of the faces are made conceptually different

than the edges of the polyhedron.

Each face-edge is paired with another face-edge such that their respective faces are direct neighbours. Together they form one edge of the polyhedron. By convention, the edges are all sorted counter-clockwise in each face, hence each pair of edges are opposite in orientation, cover the same vertices and are consequently of equal size.

The *shadow* of a face with respect to one of its edges $e$ is defined as the face of the paired edge of $e$. In effect it is its neighbour face on the other side of $e$.

In the "cut open" polyhedron shortest paths are straight lines, not geodesics. Building these lines is achieved by *unfolding* two joint faces around their common edge to make them coplanar. Rather than actually unfolding all the faces from the source to the current face being processed, the *image* of the source is kept along with its *projection* on each successive edge.

A *projection* is the section of an edge accessible from an image of the source in neighbour faces. Consequently, a shortest path to any point of the surface falls completely into an area delimited by successive projections. (That is, no shortest path gets out of such an area).

### 2.2.2 Naive algorithm

Chen & Han [4] proposed a first, naive algorithm presented in figure 1 in order to explain the basic strategy followed. The idea is to build a tree rooted on the source point $S$. Each node of this tree keeps a reference to an edge $e$, an image $I$ of the source in the face of $e$ and the projection of $I$ on $e$, noted $proj_e^I$.

The returned tree in itself is not convenient to compute distances or to build actual paths, but it holds all the necessary information to compute them. One can backtrack a path from a leaf up to the root using the projections.

### 2.2.3 Efficient algorithm

The previous algorithm is correct but it has an exponential running time due to the fact that different paths may overlap each other. Overlapping paths are a consequence of projections falling on an angles rather than on single, flat edges. In this situation, two projections are created, hence two paths.

Chen and Han proved that in the situation when more than one sequence cover the same angle, only one of them needs to have (at most) two children. Otherwise, some of the children's paths are redundant. This observation is powerful since it bounds the number of subtrees at each vertex, making the tree sparser by pruning away useless branches as soon as they appear. The complexity in space falls from exponential to linear in the number of vertices.

However, the inner nodes of the tree must be discarded and only the leaves are kept in order to achieve such a space complexity. For the sake of simplicity, these nodes are kept in the implementation to ease the backtracking phase.

In practice, when the source is projected on an angle $\alpha$, two projections are built to

**Input:** $S$: The source, the starting point of the paths
$N = \{F_1, F_2, \dots F_n\}$: The net, a list of triples of edges
$L = \{\}$: Current leaves
$T = (S, \emptyset, \emptyset)$: The tree
**for** $e \in F_S$ **do**
    // $F_S$ is the source's face
    $I = S$: The image of $S$ (trivial here)
    $proj_e^I = e$: The projection of $I_0$ on $e$ (trivial here)
    addChild$(T, (I, proj_e^I, e))$
**end**
**for** $i \in \{1, 2, \dots n\}$ **do**
    $L' = \{\}$
    **for** $l \in L$ **do**
        /* For each path remaining open (*i.e.* non-dead-end) */
        $F =$ shadow$(e_l)$: The current face, shadow of $e_l$
        $I' =$ unfold$(I_l, e_l)$: The image of $S$ in $F$ around $e_l$
        $\{e_1, e_2, e_l\} = F$: $e_1$ and $e_2$ are the opposite edges of $e_l$ in $F$
        **for** $e \in \{e_1, e_2\}$ **do**
            **if** $proj_e^{I'} \neq \emptyset$ **then**
                $l' = (I', proj_e^{I'})$
                addChild$(l, l')$
                $L' = L' \cup \{l'\}$
            **end**
        **end**
    **end**
    $L = L'$
**end**
**return** $T$

Figure 1: Chen & Han, naive algorithm

cover $\alpha$. This results in a node having two children, one for each projection. This node is said to *occupy* $\alpha$.

If a newly created node is found to cover an angle $\alpha$ that is already occupied, then the distances between their respective images and the vertex on which lies $\alpha$ are compared. The node being the closest to $\alpha$ is the one that should occupy it, hence keeping both its children. The other node does no longer need two children and keeps the only one whose projection covers an otherwise uncovered section of its supporting edge. In the special case of a tie, no node has two children.

The algorithm 2 shows the core loop of the algorithm. The other parts are left unchanged.

**for** $l \in L$ **do**
  $F = \texttt{shadow}(e_l)$: The current face, shadow of $e_l$
  $I' = \texttt{unfold}(I_l, e_l)$: The image of $S$ in $F$ around $e_l$
  $\{e_1, e_2, e_l\} = F$: $e_1$ and $e_2$ are the opposite edges of $e_l$ in $F$
  $\alpha = \texttt{angle}(e_1, e_2)$
  $o = \texttt{occupyingAngle}(\alpha)$: The node $o$ currently occupying $\alpha$
  **if** $proj_{e_1}^{I'} = \emptyset$ *or* $proj_{e_2}^{I'} = \emptyset$ *or* $o = \emptyset$ **then**
    $l'_{1,2} = (I', proj_{e_1,e_2}^{I'})$: Where the projections are non-empty
    $\texttt{addChild}(l, l'_{1,2})$
    **if** $l$ *has two children* **then** $\texttt{occupy}(l, \alpha)$: The angle $\alpha$ is now occupied by $l$
  **else**
    **if** $|I', \alpha| < |I_o, \alpha|$ **then**
      $l'_{1,2} = (I', proj_{e_1,e_2}^{I'})$: Where the projections are non-empty
      $\texttt{addChild}(l, l'_{1,2})$
      $\texttt{deleteRedundantChild}(o)$
      $\texttt{occupy}(l, \alpha)$: The angle $\alpha$ is now occupied by $l$
    **else**
      $l' = (I', proj_{e_{1:2}}^{I'})$: Where the projection is the non-redundant one
      $\texttt{addChild}(l, l')$
      **if** $|I', \alpha| = |I_o, \alpha|$ **then** $\texttt{deleteRedundantChild}(o)$
    **end**
  **end**
**end**

Figure 2: Chen & Han, efficient algorithm (core loop)

# 3 Bounds on the number of egde-to-edge gluings of squares

In this section, we prove that the number of edge-to-edge gluings of $n$ squares is polynomial in $n$. Note that the layout of these theorems allow to develop a polynomial algorithm to list the nets, one just needs to follow all the cases that we describe, and also estimate how much output this algorithm will produce.

**Theorem 1.** *There are $O\left(n^{36}\right)$ edge-to-edge gluings of at most $n$ squares that satisfy Alexandrov's conditions.*

*Proof.* Since we're considering gluings that satisfy Alexandrov's conditions, there is a polyhedron corresponding to each of them. Consider this polyhedron and triangulate it. Since this polyhedron is glued from squares, it is possible to draw each of its faces on square grid, the vertices of the face being also the vertices of the grid.

What is more, an edge shared by two faces must look the same on the drawings of these faces; an example can be seen in Figure 3. That means, it must have the same length and the tilt angle that is the same or differing by $\frac{\pi}{2}$.

The graph of the polyhedron is planar and has at most 8 vertices. Due to Euler's

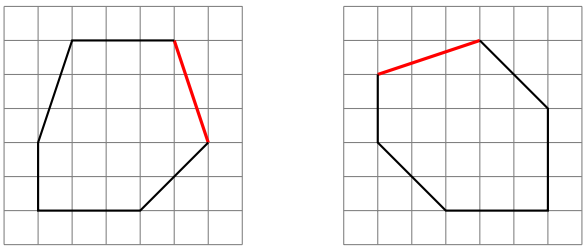


Figure 3: Highlighted edge looks the same on the drawings of two faces it belongs to

formula, it can have at most 18 edges. For each edge, we can choose its projections on $x$ and $y$ axes once it is drawn on the grid. Since the edge is a part of a flat face, all the squares that intersect the edge are distinct. There is at most $n$ of them, which yields that both projections are at most $n$, so there is at most $n^2$ ways to choose the edge.

Once the projections of the edges are known, let us draw the faces on the grid. At every vertex, there is at most two ways to place the next edge, such that the convexity of the face is preserved, those differ by $\frac{\pi}{2}$, see Figure 4 for an example.

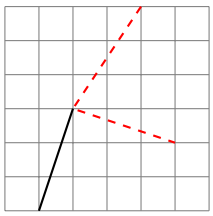


Figure 4: There are two ways to place each edge preserving convexity of the face

This adds at most $2^{2\cdot 18}$ ways to draw the net once the edges are known, which gives the total of at most $(2n)^{36}$ gluings. □

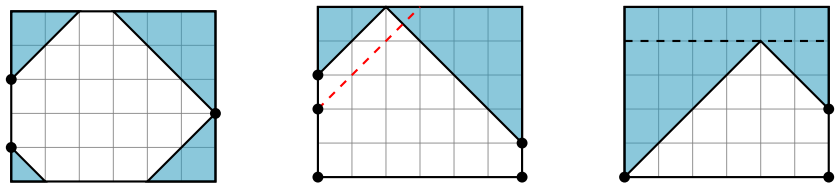
**Theorem 2.** *There are* $\Omega\left(n^{\frac{5}{2}}\right)$ *edge-to-edge gluings of at most* $n$ *squares that satisfy Alexandrov's conditions.*

*Proof.* We will construct a family of $\Omega\left(n^{\frac{5}{2}}\right)$ distinct polyhedra that can be glued from squares edge-to-edge. Those will be doubly-covered convex polygons whose edges are either parallel to the axes or inclined by $\frac{\pi}{4}$, and vertices are vertices of the grid.

Consider a square $\left\lfloor\frac{\sqrt{n}}{2}\right\rfloor \times \left\lfloor\frac{\sqrt{n}}{2}\right\rfloor$ drawn on the square grid. If a polygon is drawn inside this square, and it satisfies the conditions of the previous paragraph, then doubly covered version of this polygon can be glued from squares and consists of at most $n$ squares.

The polygons will be constructed the following way: we will take a rectangle and cut it angles off. More formally, we will pick two points on either of short sides of the rectangle and shoot lines inclined by $\frac{\pi}{4}$ from them. The points can correspond, in this case there is one vertex of the polyhedron on the corresponding side of the rectangle. An example can be seen in Figure 5a: the points we picked are marked, and the areas that are cut off are highlighted.

Define the length of the longer side of the rectangle by $a$, and length of the shorter



(a) (b) (c)

Figure 5: (a) An example of a polygon produced by cutting angles of a rectangle. (b) Some pairs of points on short sides do not produce valid polygons. (c) A polygon can be obtained by cutting angles of several different rectangles.

side by $b$. There number of ways to choose a pair of points on each of shorter sides is

$$\left(\frac{b(b+1)}{2}\right)^2.$$

However, some sets of points do not produce valid polygons, since the rays shot from them intersect not at a vertex of the grid. An example can be seen in Figure 5b: red dashed line intersects with the line from the right side in the middle of a cell. Still, at least half the pairs of upper points on the sides are valid: for each invalid pair, the pair where left point is raised by 1 is valid. We get that the number of ways to select a valid set of points is

$$\left(\frac{b(b+1)}{2}\right)^2 \Big/ 4.$$

Note that each polyhedron has at least one horizontal edge, since otherwise $a$ must be less than $b$: the sum of segments cut off horizontal edges and the sum of segments cut off vertical edges are equal. However, on the other side there may not be a horizontal edge, see Figure 5c. In this case, it is not certain from which rectangle the polygon is cut: in Figure 5c it can be either $6 \times 4$ or $6 \times 5$.

However, one polygon can be cut from at most $a$ rectangles, since we know its horizontal dimension, and at least one horizontal side is fixed. Thus, the number of rectangles we get for a given $a$ is at least

$$\sum_{b=1}^{a} \frac{b^2(b+1)^2}{16a} = \Omega\left(a^4\right).$$

Recall that $a$ can vary from 1 to $\sqrt{n}/2$, the number of polygons we can produce is

$$\Omega\left(\sum_{a=1}^{\sqrt{n}/2} a^4\right) = \Omega\left(n^{\frac{5}{2}}\right).$$

□

# 4 Description of practical results

## 4.1 Listing valid gluings

The listing procedure is given a net in advance in a form of DCEL. Then it processes this list to obtain the data that will be needed for the checks: how many vertices there are,

to what faces a vertex belongs and at which position it is in that vertex.

After this initialisation stage the procedure takes arbitrary coordinates for each vertex in each face and performs the following checks in the following order:

1) Each face has at least two vertices on the border of the $n \times n$ square, this is to avoid repeated listing faces that differ by a shift.

2) All the turns took by the edges of a single face are in the positive direction, and the sum of the exterior angles is at most $2\pi$, which corresponds to each face being a convex polygon.

3) If an edge belogs to two faces, than in those faces it has the same length and the tilt angle that is the same or differing by $\frac{\pi}{2}$.

4) Sum of angles at every vertex is at most $2\pi$.

**Theorem 3.** *Running time of the implemented algorithm is $O(n^{72})$.*

*Proof.* Once the coordinates of each vertex of each face is known, the check if the gluing is valid takes $O(1)$ time. For every vertex we need to select two coordinates, which gives $O(n^2)$ per vertex per face.

Each vertex appears as many times as many faces it belongs to, the number of those faces is equal to the degree of the vertex. Thus, we get

$$O\left(n^{2\cdot\sum\limits_{v}\deg(v)}\right) = O\left(n^{4|E|}\right) = O\left(n^{72}\right).$$

□

As an attempt to speed up the running time, the unicalization of the list of vertices was performed. Originally, if a vertex appears in $d$ faces, it will appear in the list of vertices exactly $d$ times.

Each vertex $v$ is represented by a list $[i_1, x_{i_1}, i_1, x_{i_2} \ldots]$, where $i_1, i_2, \ldots$ are faces to which $v$ belongs, and $x_{i_1}, x_{i_2}, \ldots$ are the positions of $v$ in these faces respectively. The faces are traversed in clockwise order around $v$.

As a modification of the program, each list was rotated in a way that the face with the smallest index takes the first position. Then the duplicates were removed from the multi-dimensional list of vertices, the code for this exact procedure was sourced from `stackoverflow.com`.

After the program achieved representations of the nets in numerical form, the graphic output was configured for an user to see the polygons and the way they are glued. TikZ code was selected as a platform for the output, drawings of several nets can be seen in **Appendix**.

# 5 Discussion

Out our current machinery it takes the program roughly 30 seconds to analyze 100 million nets, which means all nets consisting of two quadrilaterals fitting inside $4 \times 4$ field can be checked in 13 hours, but it will take 9'000'000 years to check all the nets consisting of 5 triangles. So, an implementation of a faster algorithm is needed to perform the check in feasible time.

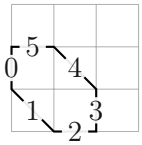
Theorem 1 suggests that there is a faster algorithm, but the degree is still huge, and it is harder to implement.

What is still to be implemented is the check of isomorphism of $8 \times 8$ matrices, it is however an insignificant $O(1)$ subroutine.

# Appendix: examples of valid nets

### Nets consisting of two hexagons

The polygons in these nets represent ones that we counted while proving Theorem 2: those are rectangles with some angles cut.



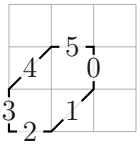
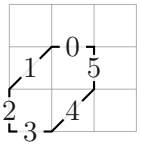




Figure 6: Net 0



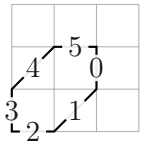
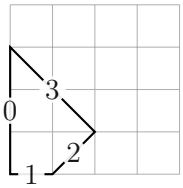




Figure 7: Net 1879605

### Nets consisting of two quadrilaterals

Some rather surprising nets can be found here. While the net in Figure 8 clearly corresponds to a doubly-covered quadrilateral, for other nets it is completely unclear which polyhedron is obtain after gluing them together.



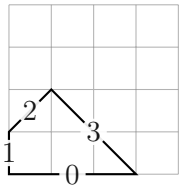
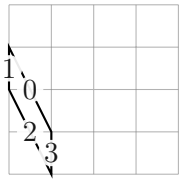




Figure 8: Net 49637490



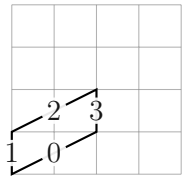




Figure 9: Net 60877856

### Nets of a four-sided pyramid

We were trying to find a non-trivial way to glue a four-sided pyramid from squares, however the only polyhedron we could find complying with this net is a doubly-covered square, see Figure 14.

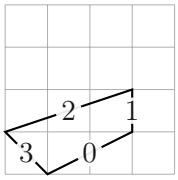
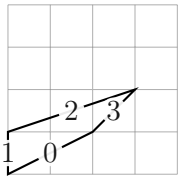



Figure 10: Net 70707676



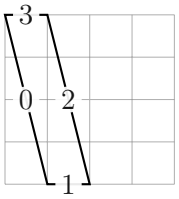
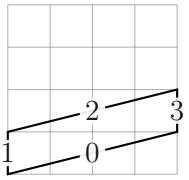




Figure 11: Net 81497520



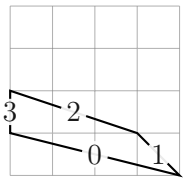
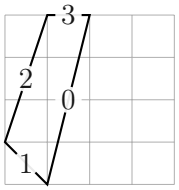




Figure 12: Net 103500700



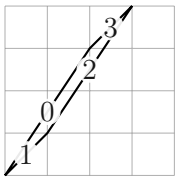
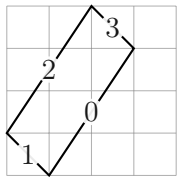




Figure 13: Net 111905612




### Net of a cube

Another net worth noting is an unusual net of cube shown in Figure 15. This way to glue the cube is rather non-trivial, and the resulting cube can be seen in Figure 16.

## References

[1] Alexandr Alexandrov. *Convex Polyhedra*. Springer-Verlag, Berlin, 2005.

[2] E. Arseneva, S. Langerman, and B. Zolotov. A complete list of all convex shapes made by gluing regular pentagons. In *XVIII Spanish Meeting on Computational Geometry*, page 1–4, Girona, Spain, 2019.

[3] Elena Arseneva and Stefan Langerman. Which Convex Polyhedra Can Be Made by Gluing Regular Hexagons? *Graphs and Combinatorics*, page 1–7, 2019.

[4] Jindong Chen and Yijie Han. Shortest paths on a polyhedron. In *6-th annual symposium on Computational geometry*, page 360–369, Berkley, California, USA, June 1990. SCG '90.

[5] Erik Demaine, Martin Demaine, Anna Lubiw, and Joseph O'Rourke. Enumerating foldings and unfoldings between polygons and polytopes. *Graphs and Combinatorics*, 18(1):93–104, 2002.

[6] Erik Demaine, Martin Demaine, Anna Lubiw, Joseph O'Rourke, and Irena Pashchenko. Metamorphosis of the cube. In *Proc. SOCG*, pages 409–410. ACM, 1999.

[7] Erik Demaine and Joseph O'Rourke. *Geometric folding algorithms*. Cambridge University Press, 2007.

[8] David Eppstein, Michael J Bannister, William E Devanny, and Michael T Goodrich. The Galois complexity of graph drawing: Why numerical solutions are ubiquitous for

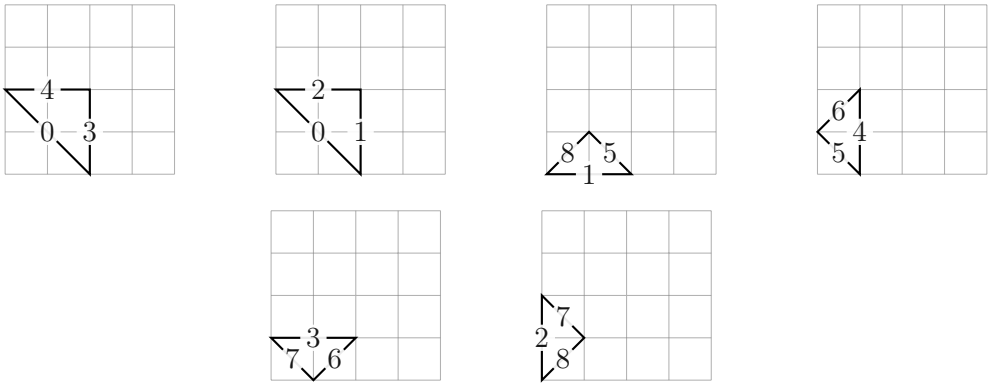



Figure 14: Net 15780252

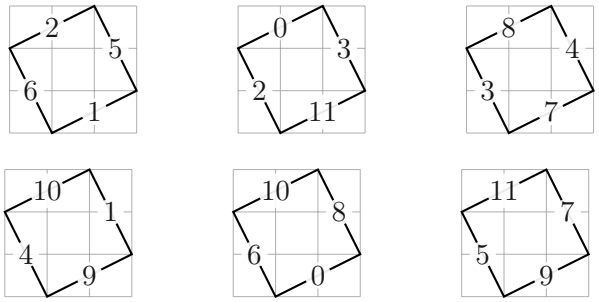



Figure 15: Net 22695

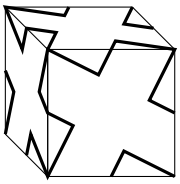


Figure 16: The way the cube is glued from the net on Figure 15

force-directed, spectral, and circle packing drawings. In *International Symposium on Graph Drawing*, pages 149–161. Springer, 2014.

[9] Daniel M Kane, Gregory N Price, and Erik D Demaine. A Pseudopolynomial Algorithm for Alexandrov's Theorem. In *WADS*, pages 435–446. Springer, 2009.

[10] Anna Lubiw and Joseph O'Rourke. When can a polygon fold to a polytope?, 1996.

[11] Boris Zolotov. Algorithmic Aspects of Alexandrov's Uniqueness Theorem. *Bachelor's Thesis, St. Petersburg State University*, 2019.