# Edge Unfoldings of Platonic Solids Never Overlap

Takashi Horiyama* Wataru Shoji*

**Abstract**

Is every edge unfolding of every Platonic solid overlap-free? The answer is yes. In other words, if we develop a Platonic solid by cutting along its edges, we always obtain a flat nonoverlapping simple polygon.

We also give self-overlapping general unfoldings of Platonic solids other than the tetrahedron (i.e., a cube, an octahedron, a dodecahedron, and an icosahedron), and edge unfoldings of some Archimedean solids: a truncated icosahedron, a truncated dodecahedron, a rhombicosidodecahedron, and a truncated icosidodecahedron.

## 1 Introduction

"Does every convex polyhedron have a nonoverlapping edge unfolding?" An unfolding (also called a general unfolding) of a polyhedron is a simple polygon obtained by cutting the surface of the polyhedron and unfolding it into a plane. For an edge unfolding, only cutting along the edges is allowed. The origin of unfoldings of a polyhedron goes back to the 16th century: In 1525, Albrecht Dürer, a painter and a mathematician, published a book entitled "Unterweysung der Messung mit dem Zirkel un Richtscheyt in Linien Ebnen uhnd Gantzen Corporen" [11]. In this book, he gave edge unfoldings of Platonic solids (also called regular convex polyhedra) and Archimedean solids (also called semi-regular convex polyhedra). There is no evidence that Dürer was aware of the question, but he seems to have some insight to the question based on his unfoldings [10]. The first explicit statement was given by Shephard in 1975 [25].

Although it was believed that every edge unfolding of a convex polyhedron never overlaps, some unfortunate cut may lead to overlapping unfoldings [9, 10, 20, 22]. If we relax the restriction of convexity, there exist non-convex polyhedra whose every edge unfolding is self-overlapping [4, 12]. If we allow general unfolding, there are two techniques for unfolding any convex polyhedron to a simple polygon [3, 21, 24], i.e., any convex polyhedron has at least one general unfolding. In [23], the probability of overlap is investigated for a random unfolding of a random polyhedron constructed using random points on a sphere.

*Graduate School of Science and Engineering, Saitama University, horiyama@al.ics.saitama-u.ac.jp, s10mm309@mail.saitama-u.ac.jp.

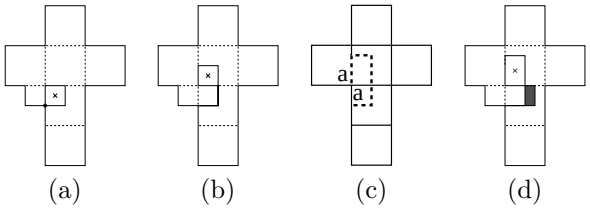


Figure 1: Overlapping general unfoldings of a cube.

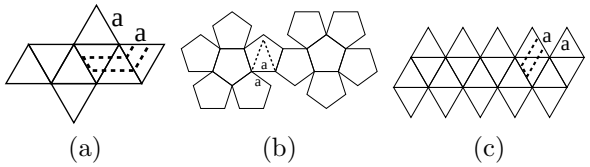


Figure 2: Overlapping general unfoldings of an octahedron, a dodecahedron, and an icosahedron.

In this paper, we consider the problem from another point of view. What happens if our polyhedron is more restricted and has a regular structure? "Are there any overlapping general unfoldings for Platonic solids?" Although this seems to be a naïve question at a first glance, we have the following two interesting observations with slightly relaxed conditions.

First, let us consider the case with general unfolding. As for a tetrahedron, any unfolding is a fundamental domain of tiling [2], i.e., any unfolding can tile the plane. This statement implicitly says that any general unfolding never overlaps. Surprisingly, for other Platonic solids, we found overlapping unfoldings: Figures 1 (a) and (b) are unfoldings of a cube that overlap in a point and a line, respectively. If we cut along the dotted line in Figure 1 (c), and glue the edges labeled $a$, we obtain an overlapping unfolding in Figure 1 (d). (Gray hatch indicates the overlap.) We can find similar overlapping unfoldings for an octahedron, a dodecahedron, and an icosahedron, respectively, in Figure 2.

Next, let us consider the case with Archimedean solids. It is known that a snub dodecahedron has an overlapping edge unfolding [9]. We also found overlapping edge unfoldings of a truncated icosahedron, a truncated dodecahedron, a rhombicosidodecahedron, and a truncated icosidodecahedron. By cutting along the bold lines of the polyhedra in Figure 3, we obtain their overlapping edge unfoldings.

As a result of the above observations, we will focus on the case for edge unfoldings: "Is every edge unfolding

of every Platonic solid overlap-free?" In other words, "Are there any overlapping edge unfoldings for Platonic solids?" As for a tetrahedron, a cube, and an octahedron, they have 2, 11, and 11 edge unfoldings [15], respectively, and we can check all of them are overlap-free by drawing them one by one. For a long time, it was believed that the same situation holds for a dodecahedron and an icosahedron. We solve this problem and say that it is correct.

**Theorem 1 (Main result)** *If we unfold a Platonic solid by cutting along its edges, we always obtain a flat nonoverlapping simple polygon.*

We solve the problem by enumerating all edge unfoldings, and check whether they are overlapping or not. It is known that a dodecahedron and an icosahedron have 43,380 edge unfoldings [6, 13]. Note that they are dual to each other. Our contribution is to strengthen this result by making a catalogue of edge unfoldings for Platonic solids. For each pair of non-neighboring faces in the unfoldings, we check whether their circumscribed circles overlap or not. Since there are no overlap, we confirm the claim that has been believed for a long time.

Our contribution also includes a proposal of enumeration algorithms by binary decision diagrams (BDDs) [1, 7] for solving problems in computational geometry. A BDD is a directed acyclic graph representing a Boolean function, and can be considered as a variant of a decision tree. By restricting the order of variable appearance and by sharing isomorphic subgraphs, BDDs have the following useful properties: (1) When an ordering of variables is specified, a BDD has the unique reduced canonical form for each Boolean function. (2) Many Boolean functions appearing in practice can be compactly represented. (3) When a BDD is given, satisfiability and tautology of the represented function can be easily checked in constant time. (4) There are efficient algorithms for many other Boolean operations on BDDs. As a result of these properties, BDDs (and its variants) are used for various practical applications, especially in computer-aided design and verification of digital systems (see e.g., [8, 17, 26]). Recently, BDDs are widely used in various fields (see e.g., [14, 19]). Knuth devoted notable space in "The Art of Computer Programming" with BDDs [16]. BDDs are regarded as a succinct data structure with efficient manipulation algorithms.

The rest of this paper is organized as follows. The next section gives fundamental concepts on BDDs. We propose algorithms for enumerating edge unfoldings and checking whether they are overlap-free or not in Section 3, and their results are given in Sections 4.

## 2 Binary Decision Diagrams

A binary decision diagram (BDD) is a directed acyclic graph that represents a Boolean function. It has two

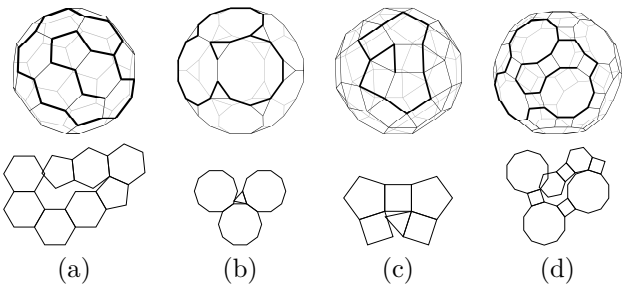


Figure 3: Overlapping unfoldings of a truncated icosahedron, a truncated dodecahedron, a rhombicosidodecahedron, and a truncated icosidodecahedron.

sink nodes 0 and 1, called the 0-node and the 1-node, respectively (which are together called the constant nodes). Other nodes are called variable nodes, and each variable node $v$ is labeled by one of the variables $x_1, x_2, \ldots, x_n$. Let $var(v)$ denote the label of node $v$. Each variable node has exactly two outgoing edges, called 0-edge and 1-edge, respectively. One of the variable nodes becomes the unique source node, which is called the root node. Let $X = \{x_1, x_2, \ldots, x_n\}$ denote the set of $n$ variables. A variable ordering is a total ordering $(x_{\pi(n)}, x_{\pi(n-1)}, \ldots, x_{\pi(1)})$, associated with each BDD, where $\pi$ is a permutation $\{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$. The level of a variable $x_{\pi(i)}$ is defined to be $i$. Similarly, the level of a node $v$ is defined by its label; if node $v$ has label $x_{\pi(i)}$, its level is defined to be $i$. That is, the root node is in level $n$ and has label $x_{\pi(n)}$, the nodes in level $n-1$ have label $x_{\pi(n-1)}$ and so on. The level of the constant nodes is defined to be 0. On every path from the root node to a constant node in an BDD, each variable appears at most once in the decreasing order of their levels. The size of a BDD is the number of nodes in it.

Every node $v$ of a BDD represents a Boolean function $f_v$, defined by the subgraph consisting of those edges and nodes reachable from $v$. If node $v$ is a constant node, $f_v$ equals to its label. If node $v$ is a variable node, $f_v$ is defined as $\overline{var(v)} f_{0\text{-}succ(v)} \vee var(v) f_{1\text{-}succ(v)}$ by Shannon's expansion, where 0-$succ(v)$ and 1-$succ(v)$, respectively, denote the nodes pointed by the 0-edge and the 1-edge from node $v$. The function $f$ represented by a BDD is the one represented by the root node. When two nodes $u$ and $v$ in a BDD represent the same function, and their levels are the same, they are called equivalent. A node whose 0-edge and 1-edge both point to the same node is called redundant. A BDD which has no mutually equivalent nodes and no redundant nodes is reduced. In the following, we assume that all BDDs are reduced.

An assignment to variables in $X$ can be regarded as a subset $S \subseteq X$, and a Boolean function $f$ can be regarded as a family $\mathcal{F} \subseteq 2^X$. For example, an assignment $(x_3, x_2, x_1) = (1, 1, 0)$ can be regarded as a set

$\{x_3, x_2\}$, and $x_1(x_2\overline{x}_3 \vee \overline{x}_2 x_3)$ can be regarded as a family $\{\{x_2, x_1\}, \{x_3, x_1\}\}$. By using BDDs for representing families of a set, we can use Boolean operations on BDDs as a family algebra (see e.g., [16]), or set operations. Later in this paper, we identify a Boolean function with its corresponding family $\mathcal{F}$, unless confusion arises.

For solving a constraint satisfaction problem, all we have to do is to interpret the restrictions of the problem as a form of Boolean functions, and represent them by BDDs. By applying AND operation to the BDDs, we can obtain the BDD representing the solutions satisfying all of the restrictions. Once such BDD is obtained, paths from the root node to the 1-node correspond to satisfying assignments for its function. Thus, we can enumerate all solutions by traversing the BDD.

## 3 Algorithms for Enumerating and Checking Edge Unfoldings

By the following three steps, we enumerate edge unfoldings of Platonic solids and check whether they are overlap-free: (1) We represent the constraints for edge unfoldings as BDDs. (2) We eliminate mutually equivalent unfoldings. (3) We check whether they are overlap-free or not. We propose algorithms for these subproblems, which are applicable to any of the Platonic solids. Later in this paper, we denote $n$ and $m$ as the number of vertices and edges of a Platonic solid, respectively.

### 3.1 Enumeration of Edge Unfoldings

We start with the following lemma that gives a good insight for edge unfoldings.

**Lemma 2 (See [10, Lemma 22.1.1])** *The cut edges of an edge unfolding of a convex polyhedron form a spanning tree of the 1-skeleton (i.e., the graph formed by the vertices and the edges) of the polyhedron.*

This lemma implies two characterizations of edge unfoldings, and we propose two algorithms according to these characterizations. The first characterization is that a set $S$ of cut edges gives an edge unfolding if and only if (1) $S$ consists of exactly $n-1$ edges and (2) no edges in $S$ form a cycle. For interpreting these constraints as Boolean functions, we use $m$ Boolean variables $x_1, x_2, \ldots, x_m$ representing whether edges are cut or not: $x_i = 1$ if its corresponding edge $e_i$ is cut, otherwise $x_i = 0$.

Condition (1) is represented as a Boolean function that outputs 1 if and only if exactly $n-1$ out of $m$ variables are 1's. Such function is one of symmetric functions, whose BDD is of size $O(m^2)$ [16]. Figure 4 shows a BDD of a function that outputs 1 if and only if 3 out of 6 variables are 1's. The left-most column of the variable nodes implies that no 1's have been received yet.

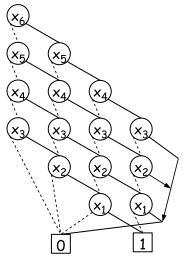


Figure 4: A BDD representing that exactly 3 out of 6 variables are 1's.

Similarly, the four columns in Figure 4 represent that we have received no 1's, one 1, two 1's, and three 1's, respectively. We call this BDD construction procedure as Choose$(n-1, m)$.

As for Condition (2), we first construct a BDD for a set of cycles, and then construct a BDD for prohibiting cycles. For constructing a BDD for cycles, we begin a set of edges in a face. Then, we repeat adding a new face and constructing cycle with the edges of the face. Figure 5 is the detail of this idea, and Figure 6 illustrates the first three iterations of Step 2.

In Procedure EnumerateCycles, we use $f_{\text{cycle}}$ to denote the obtained set of cycles. In Step 1, $f_{\text{cycle}}$ is set to be empty. In the first iteration of Step 2, we pick face $F_1$, and add a cycle with its edges. (The cycle is illustrated with a bold line in Figure 6 (a).) The cycle (more precisely, the set of edges in the cycle) is set to $f_1$. $f_2$ is empty since $f_{\text{cycle}}$ is empty. Now, the family $f_{\text{cycle}}$ contains the cycle of face $F_1$. In the second iteration, we pick face $F_2$, and its corresponding cycle is set to $f_1$ (see Figure 6 (b)). By combining the edges of $F_2$ with the already obtained cycle (the cycle in Figure 6 (a)), we can obtain a new cycle (i.e., the cycle contains the edges of $F_1$ and $F_2$). More precisely, if edge $x_j$ of $F_2$ is not in the already obtained cycle, the edge exists in the new cycle. Otherwise, the edge does not exist in the new cycle. By adding the above two cycles, $f_{\text{cycle}}$ becomes a family of the cycles in Figures 6 (a) and (b). In the following iterations, Step 2-2 combines the edges of $F_i$ with already obtained cycles. (Although $f_{\text{cycle}}$ may contain a set of edges that consists of two (or more) cycles, there is no influence on the next procedure, i.e., the construction of a BDD for prohibiting cycles.) In Step 3, we omit an empty set of edges from $f_{\text{cycle}}$. Note that the empty set is obtained as $f_{\text{empty}} := \bigwedge_{x_j \in X} \overline{x_j}$ and the set difference is obtained by $f_{\text{cycle}} \wedge \overline{f_{\text{empty}}}$.

Now, we have a family of cycles and will construct a BDD for prohibiting cycles. The complement of $f_{\text{cycle}}$ is not sufficient for this task. We should prohibit a set $S$ ($\subseteq X$) of edges if $S$ is in the monotone extension [5] of $f_{\text{cycle}}$, where the monotone extension of a family $\mathcal{F}$ of sets is a family $\{\, T \mid \text{there exists a set } T' \in \mathcal{F} \text{ satisfying } T' \subseteq T\}$. Procedure MonotoneExtension in Figure 7 construct a BDD of the monotone

**Procedure EnumerateCycles**

**Input:** A Polyhedron.
**Output:** A BDD representing a family of cycles in the give Polyhedron.

**Step 1 (initialize).** $f_{\text{cycle}} := 0$.
**Step 2 (iterate).** For each face $F_i$, apply Steps 2-1, 2-2, and 2-3, where $F_i$ has a set of edges $E_i = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$.
**Step 2-1.** Construct a BDD of $f_1 := (\bigwedge_{x_j \in E_i} x_j) \wedge (\bigwedge_{x_j \in X \setminus E_i} \overline{x_j})$.
**Step 2-2.** Construct a BDD of $f_2$ which is obtained from the BDD of $f_{\text{cycle}}$ by exchanging the roles of 0-edges and 1-edges of the variable nodes labeled by $x_j \in E_i$.
**Step 2-3.** Construct a BDD of $f_{\text{cycle}} := f_{\text{cycle}} \vee f_1 \vee f_2$.
**Step 3.** Construct a BDD of $f_{\text{cycle}} \wedge (\bigvee_{x_j \in X} x_j)$, and output it.

Figure 5: Procedure EnumerateCycles to construct a BDD representing the set of cycles in a polyhedron.

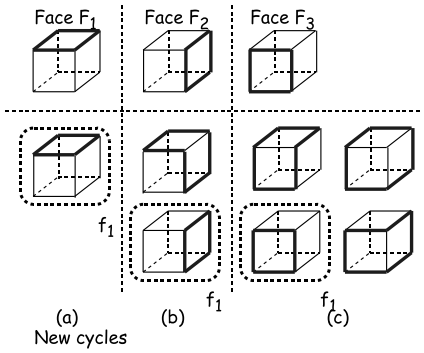


Figure 6: Example of the execution of Step 2 in Procedure EnumerateCycles.

extension of a given BDD. By combining the above procedures, we can obtain the BDD representing a family of edge unfoldings: $f_{\text{unfolding}} := \text{Choose}(n-1, m) \wedge \overline{\text{MonotoneExtension}(\text{EnumerateCycles})}$, i.e., a set difference $\text{Choose}(n-1, m) \setminus \text{MonotoneExtension}(\text{EnumerateCycles})$. We call this Algorithm 1-1.

Another characterization of edge unfoldings by Lemma 2 is as follows: A set $S$ of cut edges leads to an edge unfolding if and only if (1) $S$ consists of exactly $n-1$ edges and (2) all vertices are connected by the edges in $S$. Condition (2) is obtained by a small modification of the procedure for constructing a BDD of Hamiltonian cycles for traveling salesman problem [18]. We do not use the restriction that every vertex has exactly two edges, but use the restriction that every vertex has at least one edge. We call this Procedure EnumerateConnected. By combining this procedure with Procedure Choose, we can obtain the BDD representing a family of edge unfoldings: $f_{\text{unfolding}} := \text{Choose}(n-1, m) \wedge \text{EnumerateConnected}$. We call this Algorithm 1-2.

**Procedure MonotoneExtension**

**Input:** A BDD $G$ representing $f$. ($v$ is the root node of $G$.)
**Output:** A BDD representing the monotone extension of $f$.

**Step 1 (termination).** If $f = 0$ or $f = 1$, return $G$.
**Step 2 (recursion).** Let $G_0$ and $G_1$ be the BDDs whose root nodes are 0-$succ(v)$ and 1-$succ(v)$, respectively. Construct BDDs $G_{m0}$ and $G_{m1}$ of $f_{m0} := \text{MonotoneExtension}(G_0)$ and $f_{m1} := \text{MonotoneExtension}(G_1)$.
**Step 3 (construction).** Construct a BDD $G_{m*}$ of $f_{m*} := f_{m0} \vee f_{m1}$. Then, construct a BDD $G_m$ whose root node $v_m$ is labeled by the same variable with node $v$, 0-$succ(v_m)$ and 1-$succ(v_m)$ are the root nodes of $G_{m0}$ and $G_{m*}$, respectively. Output $G_m$.

Figure 7: Procedure MonotoneExtension to construct a BDD representing the monotone extension of $f$.

The algorithms 1-1 and 1-2 enumerate sets of cut edges. In this family, different sets of cut edges may give the same edge unfolding. We omit mutually isomorphic edge unfoldings by the lexicographic order. For example, the two sets of cut edges in Figure 8 give the same edge unfolding. The sets of cut edges in Figure 8 are represented by assignments (a) 101101001011 and (b) 110100101101, respectively, where the most significant (i.e., left most) bit corresponds to $x_{12}$ and the least significant (i.e., right most) bit corresponds to $x_1$. As (b) is lexicographically larger than (a), we omit (a). We can implement this process by manipulating BDDs.

### 3.2 Overlapping Check of Edge Unfoldings

Now, we have a family of edge unfoldings. All we have to do is to check whether each of them is overlapping-free or not. As for a tetrahedron, a cube, an octahedron,

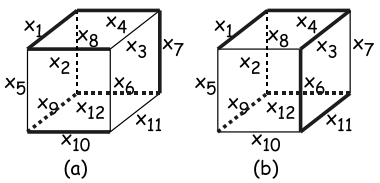


Figure 8: Isomorphic edge unfoldings.

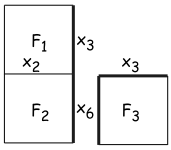


Figure 9: Neighboring faces.

and an icosahedron, the faces are equilateral triangles or squares. Thus, we can place all faces of their edge unfoldings on an equilateral triangular lattice or a square lattice. Unfortunately, the faces of a dodecahedron are pentagons, which cannot make a lattice. In this paper, we propose an algorithm that can be applied to any Platonic solid.

Given a set of cut edges, we can obtain the $x$-$y$ coordinates for the centers of the faces. For example, an unfolding of a cube has the centers on $(0,0)$, $(a\cos(\frac{3}{2}\pi), a\sin(\frac{3}{2}\pi))$, $(a\cos 0, a\sin 0)$, $(a\cos 0 + a\cos(\frac{1}{2}\pi), a\sin 0 + a\sin(\frac{1}{2}\pi))$, $(2a\cos 0, 2a\sin 0)$, $(a\cos\pi, a\sin\pi)$, where $a$ is a distance between the centers of two neighboring faces. That is, $a = 2\sin\frac{n_f-2}{2n_f}\pi$ holds, where $n_f$ is the number of vertices in a face. We assume that the circumscribed circle of a face has radius 1. We check where the distances between any two centers are larger than $a$ by Mathematica.

For any pair of faces, we check whether their circumscribed circles overlap or not. We do not apply this check to neighboring faces. We call this Algorithm 3. We emphasize here that two faces may not overlap even if their circumscribed circles overlaps. Nevertheless, as shown in the next section, there exist no overlapping circumscribed circles for any pair of the faces. In other words, there are no overlapping faces except for neighboring ones.

Figure 9 illustrates that edges $e_2$, $e_3$ and $e_6$ meet on a vertex in the original Platonic solid, and that $x_3 = x_6 = 1$ (i.e., $e_3$ and $e_6$ are cut edges) and $x_2 = 0$ (i.e., $e_2$ is not a cut edge). If a vertex has $k$ cut edges, its surrounding faces are separated into $k-1$ sets. (Recall that we have at least one cut edge for every vertex.) The faces in Figure 9 are separated into $\{F_1, F_2\}$ and $\{F_3\}$. If two faces are in the same set, they are called neighboring. In case $k = 1$, the surrounding faces are in a set, and thus, any two faces of the set are neighboring.

## 4 Experimental Results

We implemented the algorithms in Section 3 in programming language C. Table 1 gives a comparison between Algorithms 1-1 and 1-2. The computation time is measured on Intel(R) Core(TM) 2 Duo E7300 2.66GHz, 2GB memory, Ubuntu 10.04. Both algorithms give the same number of edge unfoldings: A cube and an octahedron have 384 edge unfoldings, which corresponds to the counting result in [15]. A dodecahedron and an icosahedron have 5,184,000 edge unfoldings, which corresponds to the counting result in [13]. Algorithm 1-1 runs faster than Algorithm 1-2, while both give the same number of unfoldings.

In Algorithm 1-2, we update the BDDs of $F_1, \ldots, F_n$ for $n-1$ times, and each update of $F_i$ requires $n-1$ OR operations. Thus, $O(n^3)$ OR operations are required in total. This is why the computation time grows so quickly for Algorithm 1-2. By omitting mutually isomorphic edge unfoldings, we obtain 2 unfoldings for a tetrahedron, 11 unfoldings for a cube and an octahedron, and 43,380 unfoldings for a dodecahedron and an icosahedron, respectively. Figure 10 is a partial list of enumerated edge unfoldings of a dodecahedron and an icosahedron.

Table 1: Comparison between Algorithms 1-1 and 1-2.

| | Algorithm 1-1 | | Algorithm 1-2 | |
|---|---|---|---|---|
| | #unfoldings | Time (sec) | #unfoldings | Time (sec) |
| Tetrahedron | 16 | 0.01 | 16 | 0.01 |
| Cube | 384 | 0.01 | 384 | 0.01 |
| Octahedron | 384 | 0.01 | 384 | 0.01 |
| Dodecahedron | 5,184,000 | 0.01 | 5,184,000 | 0.61 |
| Icosahedron | 5,184,000 | 0.02 | 5,184,000 | 2.91 |

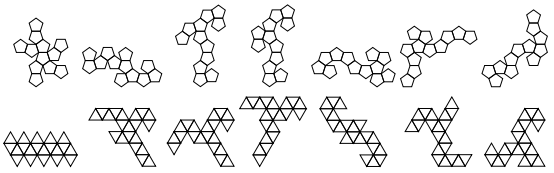

Figure 10: Partial list of edge unfoldings of a dodecahedron and an icosahedron.

Table 2 gives the total and average computation time of Algorithm 3. The average means the average computation time for an edge unfolding. For each edge unfolding, we have $O(F^2)$ pairs of faces to check whether they overlap or not, where $F$ is the number of faces. The results on average computation time are almost same among all Platonic solids. Algorithm 3 says that, in any edge unfolding, there is no pair of faces (except for neighboring faces) that have overlapping circumscribed circles. In other words, edge unfoldings of Platonic solids are simple and nonoverlapping. The list of all cut-edges for Platonic solids, their corresponding sets of inequalities in Mathematica format, and a catalogue for edge unfoldings of Platonic solids are shown in `http://www.al.ics.saitama-u.ac.jp/horiyama/research/unfolding/`.

## 5 Conclusions

We have proposed algorithms making use of BDDs for enumeration of edge unfoldings, and made a catalogue of edge unfoldings for Platonic solids. We furthermore proved that no edge unfolding of a Platonic solid overlaps. Our algorithms are applicable to Archimedean solids. It is also interesting to use ZDDs [16, 17] since it is also suitable for handling sets and families. We

Table 2: Computation Time for Algorithm 3.

| | Total Time | Average Time |
|---|---|---|
| Tetrahedron | 0.51s | 0.25s |
| Cube | 2.70s | 0.25s |
| Octahedron | 2.67s | 0.24s |
| Dodecahedron | 198m 39.16s | 0.27s |
| Icosahedron | 200m 55.15s | 0.28s |

also emphasize here that our approach for making use of BDDs is applicable to many other problems in computational geometry.

## References


[1] S. B. Akers, Binary decision diagrams, *IEEE Trans. Com.*, C-27:509–516, 1978.

[2] J. Akiyama, Tile-Makers and Semi-Tile-Makers, *Math. Assoc. America*, 114:602–609, 2007.

[3] B. Aronov and J. O'Rourke, Nonoverlap of the star unfolding, *Disc. Comp. Geom.*, 8:219–250, 1992.

[4] T. Biedl, E. D. Demaine, M. L. Demaine, A. Lubiw, J. O'Rourke, M. Overmars, S. Robbins, and S. Whitesides, Unfolding some classes of orthogonal polyhedra, *Proc. CCCG*, 70–71, 1998.

[5] E. Boros, T. Ibaraki, and K. Makino, Monotone extensions of Boolean data sets, *Proc. ALT*, LNCS 1316, 161–175, 1997.

[6] S. Bouzette, and F. Vandamme, The regular Dodecahedron and Icosahedron unfold in 43380 ways, Unpublished manuscript.

[7] R. E. Bryant, Graph-based algorithms for Boolean function manipulation, *IEEE Trans. Com.*, C-35: 677–691, 1986.

[8] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill, Sequential circuit verification using symbolic model checking, *Proc. DAC*, 46–51, 1990.

[9] H. T. Croft, K. J. Falconer, and R. K. Guy, *Unsolved Problems in Geometry*, Springer-Verlag, Reissue edition, 1995.

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

[11] A. Dürer, Unterweysung der Messung mit dem Zirkel un Richtscheyt in Linien Ebnen uhnd Gantzen Corporen, 1525.

[12] B. Grünbaum, Are your polyhedra the same as my polyhedra?, In B. Aronov, et al. (eds.), *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*, 461–488, Springer, 2003.

[13] C. Hippenmeyer, Die Anzahl der inkongruenten ebenen Netze eines regulären Ikosaeders, Elem. Math., 34:61–63, 1979.

[14] T. Horiyama and T. Ibaraki, Reasoning with ordered binary decision diagrams, *Proc. ISAAC*, LNCS 1969, 120–131, 2000.

[15] M. Jeger, Über die Anzahl der inkongruenten ebenen Netze des Würfels und des regulären Oktaeders, Elemente der Mathematik, 30:73–83, 1975.

[16] D. E. Knuth, The art of computer programming, vol. 4, fascicle 1, Bitwise tricks & techniques, Binary decision diagrams, Addison-Wesley, 2009.

[17] S. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," *Proc. DAC*, 272–277, 1993.

[18] S. Minato, Arithmetic Boolean expression manipulator using BDDs, Formal methods in system design, 10:221–242, Kluwer Academic, 1997.

[19] S. Minato and H. Arimura, Frequent Pattern Mining and Knowledge Indexing Basedon Zero-Suppressed BDDs, *Proc. KDID*, 152–169, 2006.

[20] J. Mitani and R. Uehara, Polygons Folding to Plural Incongruent Orthogonal Boxes, *Proc. CCCG*, 31–34, 2008.

[21] D. M. Mount, On folding shortest paths on convex polyhedra, Technical Report 1495, Department of Computer Science, University of Maryland, 1985.

[22] M. Namiki and K. Fukuda, Unfolding 3-dimensional convex polytopes: A package for Mathematica 1.2 or 2.0, Mathematica Notebook, University of Tokyo, 1993.

[23] C. Schevon and J. O'Rourke, A conjecture on random unfoldings, Technical report JHU-87/20, Johns Hopkins University, Baltimore, MD, 1987.

[24] M. Sharir and A. Schorr, On shortest paths in polyhedral spaces, SIAM J. Comput., 15:193–215, 1986.

[25] G. C. Shephard, Convex polytopes with convex nets, Math. Proc. Camb. Phil. Soc., 78:389–403, 1975.

[26] I. Wegener, Branching programs and binary decision diagrams, Monographs on discrete mathematics and applications, 2000.