International Journal of Computational Geometry & Applications
© World Scientific Publishing Company

ENUMERATING TRIANGULATIONS IN GENERAL DIMENSIONS

HIROSHI IMAI, TOMONARI MASADA, FUMIHIKO TAKEUCHI*
Department of Information Science, University of Tokyo
Hongo, Bunkyo-ku, Tokyo, 113-0033 Japan

and

KEIKO IMAI
Department of Information and System Engineering, Chuo University
Kasuga, Bunkyo-ku, Tokyo, 112-8851 Japan

Received 2 April 2001
Revised 15 July 2002
Communicated by B. Chazelle

We propose algorithms to enumerate (1) regular triangulations, (2) spanning regular triangulations, (3) equivalence classes of regular triangulations with respect to symmetry, and (4) all triangulations. All of the algorithms are for arbitrary points in general dimension. They work in output-size sensitive time with memory only of several times the size of a triangulation. For the enumeration of regular triangulations, we use the fact by Gel'fand, Zelevinskiĭ and Kapranov that regular triangulations correspond to the vertices of the secondary polytope. We use reverse search technique by Avis and Fukuda, its extension for enumerating equivalence classes of objects, and a reformulation of a maximal independent set enumeration algorithm. The last approach can be extended for enumeration of dissections.

Keywords: enumeration, triangulation, regular triangulation, dissection, secondary polytope, reverse search, maximal independent set.

1. Introduction

Triangulations have been one of the main topics in computational geometry and mathematics in recent years. Especially, some types of triangulations are found to bridge geometric issues and algebraic ones. Regular triangulations are of such a type.[1,2,3] For example, this subclass of triangulations has a close connection to a well-known paradigm of computer algebra, Gröbner bases, and also with theory of discriminants, hypergeometric functions, etc. (see Refs. [1,2,4,5,6,7]). Regular triangulations can be defined as a natural extension of the Delaunay triangulation and also of lexicographic triangulations, a subclass of triangulations well-known in

*Author for correspondence.

the theory of oriented matroids.

From the viewpoint of computational geometry, regular triangulations provide a good framework where many known results for triangulations of a planar point set can be generalized to higher dimensional cases. For instance, in the planar case, any pair of triangulation can be transformed to each other by a sequence of so-called Delaunay flips, but, even in three dimensional case, a Delaunay triangulation cannot always be obtained from a non-regular triangulation by Delaunay flips.[8,9] However, restricting ourselves to the class of regular triangulations, such a result has already been shown in any dimension.[1,2,3] Also, there are several works in computational geometry on regular triangulations such as Refs. [10,11].

The enumeration of (regular) triangulations is interesting from the viewpoint of computer-aided mathematical research (e.g., see Refs. [4,7,12]). Also, it is useful giving indications for algorithmic designs in computer graphics, finite element method, etc., where triangulations of three-dimensional objects are frequently used.
(1) Enumeration of regular triangulations. We first propose an output-size sensitive and work-space efficient algorithm for enumerating regular triangulations of n points in d-dimensional space. It is known that the vertices of the secondary polytope correspond to regular triangulations. Our algorithm enumerates these vertices using the reverse search technique.[13,14] The known results are summarized using the so-called volume vector, and the algorithm is described in a simple way. Its time complexity is $O(d^2 s^2 \mathrm{LP}(n-d-1, s)\#\mathcal{R})$, where s is the upper bound of the number of simplices of all dimensions contained in one regular triangulation, and $\mathrm{LP}(n-d-1, s)$ denotes the time required for solving a linear programming problem with s strict inequality constraints in $n-d-1$ variables, and $\#\mathcal{R}$ is the number of regular triangulations, which is bounded by $O(n^{(d+2)(n-d-2)})$. Its work-space complexity is $O(ds)$, which is best possible to retain one triangulation. Our time complexity is proportional to the output size $\#\mathcal{R}$, and the working space is quite small.

Two algorithms for enumerating regular triangulations have been proposed.

1. The algorithm by Billera, Filliman and Sturmfels[1] first characterizes the secondary fan dual to the secondary polytope by means of Gale transforms, and then algorithmically by applying the hyperplane arrangement algorithm in Ref. [15], the problem is shown to be solvable in $O(n^{(d+1)(n-d-2)})$ time and space. This algorithm is worst-case optimal for the so-called Lawrence polytopes which form a very restricted class. However, the reduction has a redundant part for other cases, and the number of regular triangulations may be much smaller than the complexity of the arrangement. Thus, even if a good algorithm for arrangements is available, an output-size sensitive and work-space efficient algorithm is hard to obtain along these lines.

2. An output-size sensitive algorithm is given by De Loera.[12] It is based on breadth-first search enumeration, and is implemented using `Maple`. Since it is based on breadth-first search, its work-space complexity is $\Omega(\#\mathcal{R})$, which is

prohibitively large even for small-size problems.

The algorithm presented in this paper is a refined version of Refs. [16,17,18], together with some new results for spanning triangulations, and has theoretical merits as described above (codes are available from Ref. [19]). Practical merits of this algorithm is seen in subsection 4.6. Preliminary computational results are also shown.
(2) Enumeration of spanning regular triangulations. Next, we consider regular triangulations using all points. Some regular triangulations may not use a point inside the convex hull, which may not be preferable for three-dimensional applications in computer graphics and the finite element method. Triangulations using all the points are called spanning, and an algorithm with similar complexities is given to enumerate all spanning regular triangulations. Also, the diameter of the secondary polytope, the vertices of which correspond to the regular triangulations, is shown to be $O(n^{d+2})$.
(3) Enumeration of equivalence classes of regular triangulations. As mentioned above, regular triangulations have connection to many mathematical concepts such as Gröbner bases, and in such cases a given point configuration is mostly degenerate, and furthermore has symmetric structures. Then, enumerating only a representative triangulation from each equivalence class induced by the symmetry becomes crucial, since the number of triangulations equivalent under the symmetry may become large.

De Loera's program can take this symmetry into account, and he enumerates the triangulations, all of which are regular, for the case of $\Delta_2 \times \Delta_3$ and $\Delta_2 \times \Delta_4$[12,20]. When the dimensions become larger, even the number of classes divided by symmetry becomes huge. De Loera is using breadth first search in his program, so all visited triangulations should be kept in memory, and the memory constraint becomes serious in larger cases.

We propose an algorithm to enumerate equivalence classes of objects by reverse search. And then apply this to the enumeration of equivalence classes of regular triangulations with respect to the symmetry of symmetric polytopes. Applications to products of two simplices and hypercubes are shown. The algorithm runs in output-size sensitive time, i.e., in time proportional to the number of classes, and requires memory only several times that of one triangulation. An enumeration for equivalence classes of object is also proposed in Ref. [21].
(4) Enumeration of triangulations. We finally propose an algorithm to enumerate all triangulations, regular or not, in general dimension. De Loera found a nonregular triangulation in $\Delta_3 \times \Delta_3$. Enumerating all triangulations, regular or not is of interest mathematically. Algorithms to enumerate triangulations in dimension two can be found, for example in Ref. [22]. However, for dimension higher than two, though there are some results,[23] there is no efficient algorithm to enumerate all triangulations. (Though not published, a recent algorithm[24] seems to enumerate efficiently triangulations in general dimension, or even for oriented matroids.) Our algorithm (extended from Ref. [25]) enumerates them for arbitrary configurations

of points. We characterize triangulations as a subclass of maximal independent sets of the intersection graph of the maximal dimensional simplices. We reformulate a general maximal independent set enumeration algorithm, for the case of graphs, and apply it to this intersection graph. The time complexity is proportional to the number of maximal independent sets, the objects we really enumerate. When triangulations form a proper subset of the maximal independent sets, the gap between them becomes a loss. If this gap is small, this algorithm is efficient, the first efficient one, to enumerate all triangulations. The existence of this gap is determined geometrically by the configuration of points. In two dimension this does not happen, and in three dimension, we have Schönhardt's polyhedron (cf. 10.2.1 of Ref. [26]) for example. The memory required in this algorithm is only about the size of two triangulations.

We apply this to the case of the product of two simplices. The number of the simplices, which correspond to the vertices of the intersection graph, increases exponentially with dimension, but we cope with this by using their correspondence with spanning trees of a bipartite graph, and keeping in memory one simplex, or spanning tree, at once.

An implementation of this algorithm in *Mathematica* is available from Ref. [19]. An extension of our formulation for enumerating dissections is also shown.

Outline of this paper. We begin by a brief explanation of reverse search, and then give our formulation of reverse search for equivalence classes of objects (section 2). Next, we summarize the definitions and properties of regular triangulations and the secondary polytope (section 3). We give our algorithm to enumerate regular triangulations (section 4). We also consider spanning regular triangulations, and investigate the diameter of the secondary polytope (section 5). Next, we present the algorithm for the enumeration of equivalence classes of regular triangulations, and apply it to products of simplices and hypercubes (section 6). We summarize the general maximal independent set enumeration algorithm, and show our formulation for graphs (section 7). Finally, we apply this to the enumeration of all triangulations or dissections (section 8).

2. Reverse Search

Avis and Fukuda introduced an enumeration technique called reverse search.[14] It runs in time proportional to the number of objects to be enumerated, and requires memory only of several times the size of an object. We first explain their structure for enumeration (subsection 2.1), and then show our extension for enumeration of equivalence classes of objects (subsection 2.2).

2.1. *Reverse Search*

Reverse search is a general technique for enumeration. It performs at the same output-size sensitive time as breadth first search (BFS) or depth first search (DFS), but requires memory only of twice the size of an object among those to be enumerated. BFS and DFS need output-size sensitive memory to store all reached objects. To save memory, in addition to the adjacency relation, which is necessary for BFS and DFS, parent-children relation is required for reverse search.[13,14]

First we state the adjacency and parent-children relation for reverse search, which we call here a *reverse search structure.* It defines a tree structure underlying the graph of adjacency relation.

Definition 1 (reverse search structure Ref. [14]) $(S, \delta, \mathrm{Adj}, f)$ *is a* reverse search structure *if it satisfies the followings.* (1) S *is a finite set.* (2) $\delta \in \mathbb{N}$. (3) $\mathrm{Adj} : S \times \{1, \ldots, \delta\} \to S \cup \{\emptyset\}$. *For any* $a \in S$ *and* $i, j \in \{1, \ldots, \delta\}$, (i) $\mathrm{Adj}(a, i) \neq a$ *and* (ii) *if* $\mathrm{Adj}(a, i) = \mathrm{Adj}(a, j) \neq \emptyset$ *then* $i = j$. (4) $f : S \to S$ *is the parent function:* $f(a) = a$ *or* $\mathrm{Adj}(a, i)$ *for some* i. (5) *There exists a unique root object* $r \in S$*: an object such that* $f(r) = r$*. For any other object* $a \neq r$*, there exists* $n \in \mathbb{N}$ *such that* $f^{(n)}(a) = r$.

S is the set of objects to be enumerated. The maximum degree of the adjacency graph is δ. For each object $a \in S$ the adjacency function Adj returns its indexed adjacent object, or sometimes $\emptyset$ if the object has degree less than δ. This index is for use in the enumeration algorithm. We always assume that the adjacency relation is *symmetric*: if $\mathrm{Adj}(a, i) = b$ then $\mathrm{Adj}(b, j) = a$ for some j.

The information of δ, Adj, f and r is given to the reverse search algorithm, and the algorithm returns S as its output. Actually r is not necessary, because it can be found by applying f several times to an object.

Theorem 2 (Corollary 2.3. of Ref. [14]) *For the reverse search structure in definition 1, the reverse search algorithm in Ref. [14] enumerates all objects in* S*. The time complexity is* $O(\delta\,(\mathrm{time}(\mathrm{Adj}) + \mathrm{time}(f))\,\#S)$*, where* $\mathrm{time}(\mathrm{Adj})$ *and* $\mathrm{time}(f)$ *are the time necessary to compute functions* Adj *and* f*. The memory required is twice the size of an object in* S.

2.2. *Reverse Search for Equivalence Classes*

Later, we give an algorithm to enumerate equivalence classes of regular triangulations. This is based on the enumeration of equivalence classes of objects by reverse search we propose here.

We use $\sim$ for an equivalence relation on the objects S. The equivalence class of an object a is denoted by $[a]$. By rep we denote the representative function: for any object a, $\mathrm{rep}(a) \sim a$, and for any objects a, b, $a \sim b$ if and only if $\mathrm{rep}(a) = \mathrm{rep}(b)$. The composition $(\mathrm{rep} \circ f)(a)$ denotes $\mathrm{rep}(f(a))$. The following definition ensures that we can reach the "root class" by applying $\mathrm{rep} \circ f$ successively to any (element of an) equivalence class.

Definition 3 (reverse search structure for equivalence classes) $(S, \delta, \mathrm{Adj}, f, \sim, \mathrm{rep})$ *is a* reverse search structure for equivalence classes *if*

- $(S, \delta, \mathrm{Adj}, f)$ *is a reverse search structure.*
- $\sim$ *is an equivalence relation and* rep *is a representative function on* S.

- *a adjacent to b and $c \sim a$ implies the existence of an object d adjacent to c and $d \sim b$, for any a, b and c.*

- *The root object r of the original reverse search structure is the only object with $(\mathrm{rep} \circ f)(r) = r$. For any other object $a \neq r$, there exists $n \in \mathbb{N}$ such that $(\mathrm{rep} \circ f)^{(n)}(a) = r$.*

Theorem 4 *For the reverse search structure for equivalence classes in Definition 3, we can enumerate the equivalence classes of objects by the following reverse search structure. The functions* Adj *and f in the right hand are those of the original reverse search structure as in Definition 1.*

- *$S/{\sim} = \{[a] : a \in S\}$ is the set we want to enumerate*

- *δ is the same as the original reverse search structure*

- $$\mathrm{Adj}([a], i) = \begin{cases} [\mathrm{Adj}(\mathrm{rep}(a), i)] & \textit{if } \mathrm{Adj}(\mathrm{rep}(a), i) \neq \emptyset \textit{ and } [\mathrm{Adj}(\mathrm{rep}(a), i)] \neq [\mathrm{rep}(a)] \textit{ and} \\ & \textit{if } [\mathrm{Adj}(\mathrm{rep}(a), i)] \neq [\mathrm{Adj}(\mathrm{rep}(a), j)] \textit{ for any } j < i \\ \emptyset & \textit{otherwise} \end{cases}$$

- $f([a]) = [f(\mathrm{rep}(a))]$

The time complexity is $O(\delta(\delta(\mathrm{time}(\mathrm{Adj}) + \mathrm{time}(\mathrm{rep})) + \mathrm{time}(f))\#(S/\sim))$ where time(rep) *is the time to compute the representative object of the class of an given object, and* time(Adj) *and* time(f) *is the time as in the original reverse search structure. The memory required is $\delta + 2$ times the size of an object.*

Two classes are adjacent if and only if there are adjacent objects from each of them. Any object of a class has an adjacent object in all the class-wise adjacent classes. Thus the degree of adjacency for the reverse search of equivalence classes is not larger than the degree for the original reverse search, and we can use the same δ.

The following is a special case of reverse search, given by an adjacency function and a total order on the objects S.

Definition 5 (reverse search structure with total order) $(S, \delta, Adj, <)$ *is a* reverse search structure with total order *if*

- *$(S, \delta, \mathrm{Adj})$ satisfies conditions (1) to (3) in Definition 1.*

- *$<$ is a total order on S.*

- *Only the maximum element r of the total order satisfies $\max_<(\{a \in S : a = \mathrm{Adj}(r, i)$ for some $i\} \cup \{r\}) = r$.*

Proposition 6 *A reverse search structure with total order $(S, \delta, \mathrm{Adj}, <)$ together with*

- *$f(a) = \max_<(\{b \in S : b = \mathrm{Adj}(a, i)$ for some $i\} \cup \{a\})$*

becomes a reverse search structure.

We introduce a reverse search structure for equivalence classes for this version.

Definition 7 (reverse search structure for equivalence classes with total order) $(S, \delta, \mathrm{Adj}, <, \sim)$ *is a* reverse search structure for equivalence classes with total order *if*

- $(S, \delta, \mathrm{Adj}, <)$ *is a reverse search structure with total order*
- $\sim$ *is an equivalence relation on* S*.*
- a *adjacent to* b *and* $c \sim a$ *implies the existence of an object* d *adjacent to* c *and* $d \sim b$*, for any* a*,* b *and* c*.*

Proposition 8 *The reverse search structure for equivalence classes with total order together with*

- $f(a) = \max_<(\{b \in S : b = \mathrm{Adj}(a, i) \text{ for some } i\} \cup \{a\})$
- $\mathrm{rep}(a) = \max_<([a])$

becomes a reverse search structure for equivalence classes.

3. Regular Triangulations and the Secondary Polytope

Regular triangulations form a subset of triangulations. They correspond to the vertices of a polytope, the secondary polytope, which is determined uniquely by a configuration of points. We later propose an algorithm to enumerate regular triangulations by applying a vertex enumeration method to this secondary polytope. Refer to Refs. [1,2,3,5,27] for further information on regular triangulations.

Let $\mathcal{A} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\} \subset \mathbb{R}^d$ be a configuration of points, with their convex hull $\mathrm{conv}(\mathcal{A})$ having dimension d. We are interested in triangulations of $\mathrm{conv}(\mathcal{A})$. We only consider triangulations whose vertices are among the given points $\mathcal{A}$.

Two simplices σ_i and σ_j *intersect properly* if their intersection $\sigma_i \cap \sigma_j$ is a (possibly empty) face of both simplices. This is equivalent to $\sigma_i \cap \sigma_j = \mathrm{conv}(\mathrm{vert}(\sigma_i) \cap \mathrm{vert}(\sigma_j))$, where $\mathrm{vert}(\sigma_i)$ and $\mathrm{vert}(\sigma_j)$ are the sets of vertices of σ_i and σ_j. Simplices *intersect improperly* if they are not intersecting properly.

A set of d-simplices $\{\sigma_1, \ldots, \sigma_m\}$ whose vertices are among $\mathcal{A}$ is a *triangulation* of $\mathcal{A}$ if (1) any pair of simplices σ_i, σ_j intersect properly and (2) the union of the simplices $\cup\{\sigma_1, \ldots, \sigma_m\}$ is equal to $\mathrm{conv}(\mathcal{A})$. The whole set of d-simplices is denoted by $\mathcal{S}$.

A triangulation T of $\mathcal{A}$ is *regular* if there exists a vector $\psi : \mathcal{A} \to \mathbb{R}$ having the following property. For $P = \mathrm{conv}\left\{\begin{pmatrix} \boldsymbol{a}_1 \\ \psi_1 \end{pmatrix}, \ldots, \begin{pmatrix} \boldsymbol{a}_n \\ \psi_n \end{pmatrix}\right\}$, and π the projection $\pi : \mathbb{R}^{d+1} \to \mathbb{R}^d$ with $\pi\begin{pmatrix} \boldsymbol{x} \\ x_{d+1} \end{pmatrix} = \boldsymbol{x}$, $T = \{\pi(F) : F \text{ is a lower facet of } P\}$. Here F is a *lower* facet, if for $\boldsymbol{c} \in \mathbb{R}^d$, $c_0 \in \mathbb{R}$ defining $F = \{\boldsymbol{x} \in P : \boldsymbol{cx} = c_0\}$ with $\boldsymbol{cx} \leq c_0$ being valid for P, the condition $c_0 < 0$ is satisfied. Notice that this definition admits regular triangulations which do not use some of the given points, while vertices of

$\mathrm{conv}(\mathcal{A})$ are necessarily used. Regular triangulations using all points are treated in section 5.

Let T be a triangulation of $\mathcal{A}$. The *volume vector* for T is a vector $\boldsymbol{\varphi}_T : \mathcal{A} \to \mathbb{R}$ with $\varphi_T(\boldsymbol{a}_i) = \sum_{\sigma \in T: \boldsymbol{a}_i \in \mathrm{vert}(\sigma)} \mathrm{vol}(\sigma)$, where $\mathrm{vol}(\sigma)$ is the volume and $\mathrm{vert}(\sigma)$ is the set of vertices of a d-simplex σ.

The *secondary polytope* $\Sigma(\mathcal{A})$ of a point configuration $\mathcal{A}$ is the convex hull of the points $\boldsymbol{\varphi}_T$ in $\mathbb{R}^{\mathcal{A}}$ for all triangulations T of $\mathcal{A}$.

A subset of $k+2$ points from $\mathcal{A}$ is a *circuit* if the $k+2$ points and any of the $k+1$ points have a convex hull of dimension k. The points forming a circuit have exactly two triangulations. A circuit consisiting of $k+2$ points from $\mathcal{A}$ is supported by a triangulation T of $\mathcal{A}$ (which consists of d-simplices), if for one triangulation X of the circuit (which consists of k-simplices), the d-simplices U among T having a k-simplex from X as a face comprises the product of the k-simplices X and some set Y of $(d-k-1)$-simplices: $U = X \times Y$ (which becomes $X \times \{\emptyset\} = X$ when $k = d$). In such case, the d-simplices made as the product of the other triangulation X' of the circuit and the $(d-k-1)$-simplices Y, together with $T \setminus U$ become another triangulation $T' = (T \setminus U) \cup (X' \times Y)$ of $\mathcal{A}$. The new triangulation is *transformed from T by a flip* along the circuit. In this case, T can also be transformed from T' by a flip.

Regular triangulations are known to correspond to the vertices of the secondary polytope $\Sigma(\mathcal{A})$. The vertices connected by an edge in the secondary polytope are "similar": they can be modified each other by flips.

Theorem 9 (Chapter 7. Theorem 1.7., Theorem 2.10. of Ref. [2]) *The secondary polytope $\Sigma(\mathcal{A})$ has dimension $n-d-1$, and its vertices correspond one-to-one to the volume vectors of the regular triangulations of $\mathcal{A}$. The edges are between vertices whose corresponding regular triangulations can be transformed each other by a flip.*

It should be noted that a new triangulation obtained by applying a flip to a regular triangulation is not necessarily regular. During enumeration, we visit a new triangulation from a known one using flips. Thus, we have to check the regularity for each newly obtained triangulation.

The next lemma is an implication of the upper bound theorem of convex polytopes and spheres.[28]

Lemma 10 *The number of the d-simplices and all of their faces in a triangulation of $\mathcal{A}$ is bounded by the number of the faces with the same dimension of a cyclic $(d+1)$-polytope with n vertices. Especially, the number of d-simplices is bounded from above by $O(n^{\lfloor (d+1)/2 \rfloor})$.*

For the rest, s denotes the maximum number of all the d-simplices and their faces used in a regular triangulation of $\mathcal{A}$, and s_d the maximum number of the d-simplices.

4. Enumeration of Regular Triangulations

We present an algorithm for the enumeration of regular triangulations (subsection

4.1). We use our formulation of reverse search, defined in Definition 5 and Proposition 6. We next describe the data structure for representing a regular triangulation for efficient manipulation (subsection 4.2). Then, we show that it can be checked by linear programming whether a given triangulation is regular (subsection 4.3). Also, how to obtain an initial regular triangulation is explained (subsection 4.4). The complexities achived by these treatments are given (subsection 4.5). Finally, some prelimary computational results are shown (subsection 4.6).

4.1. *Enumerating Regular Triangulations*

Triangulations are defined to be adjacent if they can be transformed by a flip (see Section 3).

Definition 11 (reverse search structure for regular triangulations) *The* reverse search structure for regular triangulations *of an arbitrary point configuration $\mathcal{A}$ is*

- $S = \{$*regular triangulation*$\}$
- $\mathrm{Adj}(T, i) =$ (*the i-th regular triangulation which can be transformed from T by a flip*)

The index i in the definition of $\mathrm{Adj}(T, i)$ is not of importance. A vector $(x_1, \ldots, x_n) \in \mathbb{R}^n$ is *smaller* than a vector $(y_1, \ldots, y_n) \in \mathbb{R}^n$ in *lexicographic order*, if for some i, $1 \leq i \leq n$, $x_i < y_i$, and $x_j = y_j$ for any $j < i$.

Definition 12 (total order on regular triangulations) *We introduce a total order on regular triangulations by comparing their volume vectors in lexicographic order.*

Since regular triangulations correspond to the vertices of the secondary polytope $\Sigma(\mathcal{A})$, and lexicographic order is same as ordering the vertices by the inner product with a vector $(N^n, N^{n-1}, \ldots, N)$ with sufficiently large N, the last condition in Definition 5 is satisfied. Thus, the reverse search structure and the total order above satisfy the conditions of reverse search structure with total order. Thus, we can enumerate all regular triangulations using Proposition 6. The reverse search tree is the traces of the simplex method from all the vertices for the secondary polytope under the above mentioned cost vector.

Theorem 13 (enumerating regular triangulations) *The structure of Definition 11 and 12 enables reverse search. The time complexity is $O(d^2 s^2 \mathrm{LP}(n-d-1, s)\#\mathcal{R})$, where s is the upper bound of the number of simplices of all dimensions contained in a regular triangulation and $\mathrm{LP}(n-d-1, s)$ is the time required to solve a linear programming problem with s strict inequalities constraints in $n-d-1$ variables, and $\mathcal{R}$ is the set of regular triangulations. The memory required is $O(ds)$.*

Proof. The degree δ, time and space complexity for Adj and f are given in Proposition 16. □

4.2. *Data Structure for a Triangulation*

We represent a simplex by the set of the indices of its vertices. For each triangulation, we keep the graph of its face poset in memory. This requires $O(ds)$ space, where s was the maximum number of simplices of all dimensions used in a regular triangulation of $\mathcal{A}$.

Besides this graph, we maintain all circuits supported by the current triangulation. Each circuit is conceptually represented by a $(k+2)$-tuple $(k \leq d)$ of the indices of the points in the circuit. For a circuit with $k+2$ points, its convex hull consists of at most $k+1$ k-simplices, and in practice we represent the $(k+2)$-tuple of points implicitly by recording those simplices. Any k-simplex belongs to at most $k+1$ of the circuits consisting of more than one simplex. And, there are no more than n circuits consisting of one simplex. Thus, the number of circuits is bounded by $O(ds)$, and also the space required for this implicit representation is $O(ds)$.

For each regular triangulation, we also maintain its volume vector.

When updating triangulations by a flip, we have to maintain these data structures. The face lattice can be updated in $O(ds)$ time. Since the computation of the volume of a simplex can be done in $O(d^3)$ time, the volume vector can be computed in $O(d^3 s)$ time. By a flip, at most $(d+1)s$ of the circuits consisting of more than one simplex are deleted or inserted to the list of the circuits. Computing all such circuits can be done in $O(d^4 s)$ time. Checking whether such circuit is supported by the triangulation can be done in $O(ds)$ per each. There are at most n circuits consisting of one simplex. Computing such circuits can be done in $O(d^4 sn)$ time. Hence, the list for a flip can be computed in $O(d^4 s^2)$ time.

When a new triangulation is computed, we have to check its regularity by solving a linear programming problem in $O(\mathrm{LP}(n-d-1, s))$ time, as described in Lemma 14 below. The time complexity to solve a linear programming problem with n variables and m constraints is denoted by $\mathrm{LP}(n, m)$. By interior point method, this takes $n^3 L$ operations, where L is the size of the input. In the sequel, we assume that the time complexity to update the data structure by a flip is dominated by $O(\mathrm{LP}(n-d-1, s))$.

A point configuration in dimension d is in *general position* if any of its $d+2$ points has a convex hull of dimension d. For such point configuration, we only have to hold the graph of the d and $(d-1)$-simplices. In this case, both the space and time complexity can be reduced.

4.3. *Checking the Regularity of a Triangulation*

In the existing literature, the regularity check is done in the dual space. We here give a simple primal approach. A triangulation of a point configuration $\mathcal{A}$ is regular if the d-simplices are the projection of the lower facets of a $(d+1)$-polytope P in $\mathbb{R}^{d+1}$. Let w_i represent the weight of $\boldsymbol{p}_i$ lifting each point to $\mathbb{R}^{d+1}$. For each $(d-1)$-face σ of the triangulation not on the boundary of $\mathrm{conv}(\mathcal{A})$, the corresponding $(d-1)$-face F of P is a lower face (i.e., its defining inequality $F = \{\boldsymbol{x} \in P : \boldsymbol{cx} = c_0\}$ with $\boldsymbol{cx} \leq c_0$ valid for P satisfies $c_0 < 0$). The $(d-1)$-face σ is shared by two d-

simplices in the triangulation. Suppose the two simplices have vertices $\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d\}$ and $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{d+1}\}$. Then, we have

$$\begin{vmatrix} 1 & \cdots & 1 & 1 \\ \boldsymbol{p}_0 & \cdots & \boldsymbol{p}_d & \boldsymbol{p}_{d+1} \\ w_0 & \cdots & w_d & w_{d+1} \end{vmatrix} \cdot \begin{vmatrix} 1 & \cdots & 1 \\ \boldsymbol{p}_0 & \cdots & \boldsymbol{p}_d \end{vmatrix} > 0. \qquad (*)$$

Lemma 14 *A given triangulation is regular if and only if there is a solution* $\boldsymbol{w}$ *satisfying* $(*)$ *for each pair of adjacent* d*-facets* $\{\boldsymbol{p}_0, \cdots, \boldsymbol{p}_d\}$ *and* $\{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_{d+1}\}$.

Proof. The "only if" part is indicated above. For the "if" part, we construct a piecewise linear function by lifting the d-simplices in the triangulation by a solution $\boldsymbol{w}$. For regularity, we need to show this function is convex. If not, two points exist on the function, with their middle points in $\mathbb{R}^{d+1}$ having values smaller than on the function. By considering the restriction of this function to a plane including the two points and parallel to the $(d+1)$-th axis, the argument reduces to the case $d = 1$, which is elementary. □

Thus, in a primal way, the regularity can be checked by linear programming. It is easy to see that for a fixed simplex we can set $w_i = 0$ for each point of the simplex without changing the existence of the solution. The number of $(d-1)$-simplices in a regular triangulation is smaller than s. It can also be bounded by $(d+1)s_d/2$. Hence, this linear programming is to check the existence of a solution to $n-d-1$ variables and at most s constraints. Denote by $\mathrm{LP}(n-d-1, s)$ the time required to solve this linear programming problem.

Proposition 15 *The regularity of a triangulation can be judged in* $\mathrm{LP}(n-d-1, s)$ *time.*

4.4. *Constructing an Initial Regular Triangulation*

Our algorithm requires a regular triangulation to start. This can be an arbitrary regular one. For conceptual simplicity and some technical merits, we may consider two candidates for the initial one. One is a regular triangulation whose volume vector is lexicographically maximum among all volume vectors. The other is the Delaunay triangulation. In the latter case, we can use an algorithm for convex hulls in Refs. [13,29,30]. Ref. [10] gives an algorithm which directly constructs a regular triangulation from an assignment of weights, while its time complexity is analyzed by means of randomized analysis since the algorithm uses the flipping operation as a primitive.

The lexicographically maximum triangulation can be computed by starting with any regular triangulation and transforming it by flips towards the lexicographic maximum along a path on the secondary polytope. When the input points in $\mathcal{A}$ are in general position, the optimal regular triangulation can be obtained simply by considering a triangulation formed by points on the convex hull boundary and $\boldsymbol{a}_1$ such that all simplices have $\boldsymbol{a}_1$ as a vertex. Such a triangulation is uniquely determined.

In any case, the time necessary for obtaining the initial regular triangulation is negligible in comparison with the time necessary for the rest of the enumeration.

4.5. *Complexities*

Proposition 16 (1) *For each vertex in the reverse search tree, there are at most $\delta = O(ds)$ adjacent triangulations.*
(2) *For a vertex in the reverse search tree, its i-th adjacent vertex can be computed in* $\mathrm{time}(\mathrm{Adj}) = O(\mathrm{LP}(n-d-1,s))$ *time and $O(ds)$ space.*
(3) *For a vertex in the reverse search tree, its parent can be computed in* $\mathrm{time}(f) = O(ds\mathrm{LP}(n-d-1,s))$ *time and $O(ds)$ space.*

Proof. (1) As in subsection 4.2, for any triangulation the number of supported circuits is bounded by $O(ds)$.
(2) This can be done by updating the triangulation along the i-th circuit and checking its regularity. The complexity is from subsection 4.2, 4.3.
(3) To find the parent, we enumerate all adjacent triangulations, check their regularity, and find the lexicographically maximum one. There are at most $O(ds)$ adjacent triangulations and each of them can be computed separately. □

4.6. *Preliminary Computational Results*

We here describe computational results for randomly generated points. Concerning the results for regularly structured point sets which are interesting from a mathematical viewpoint, see Refs. [16,17]. These results shown here are still preliminary.

Our algorithm is implemented in C language (codes are available from Ref. [19]). Experiments are done on Sun Blade100 with 1.28 GB memory. Exact arithmetic is realized by the GNU MP library for arbitrary precision integer and rational number arithmetic. Linear programming problems are solved by the simplex method with Bland's rule. The space complexity is a little more than $O(ds)$ for speeding up the computation in this implementation. Our implementation also works for degenerate inputs.

We here show the number of regular triangulations and the time used for enumeration when points are generated as random integer points in the d-cube with edge length 1000. The real computational time of enumeration per one triangulation is increasing only mildly for d or n (Table 1, Figure 1).

Our system can solve much larger cases as follows. For example, the system can partially enumerate a set of 24 degenerate points in 20 dimensions, arising from a graph, such that their regular triangulations consist of at most 306 triangles (in this case the total number of triangulations is huge and we could only enumerate part of them, and yet some useful information, such as the existence of a nonregular triangulation which can be transformed by a flip from a regular one, could be obtained from partial computational results).

Table 1. Average (minimum–maximum) number of regular triangulations enumerated (above), and time in seconds used (below) for twenty random configurations of n points in dimension d.

d	n				
	5	6	7	8	9
1	8 (8–8)	16 (16–16)	32 (32–32)	64 (64–64)	128 (128–128)
2		14.45 (14–15)	49.30 (42–57)	170.85 (132–227)	715.80 (539–887)
3			25.50 (25–29)	144.80 (133–180)	1037.90 (840–1548)
4				42.45 (41–44)	413.80 (377–495)
5					67.00 (65–69)

d	n				
	5	6	7	8	9
1	0.58 (0.54–0.75)	2.59 (1.06–3.08)	11.60 (11.02–12.07)	46.01 (35.50–49.42)	126.40 (96.00–128.00)
2		2.02 (1.81–2.70)	18.82 (13.52–24.72)	148.60 (98.7–213.51)	1370.10 (858.66–1941.70)
3			8.05 (7.16–9.86)	145.10 (116.44–203.87)	2152.04 (83.99–3280.7)
4				31.49 (26.61–36.50)	1212.54 (942.27–1816.29)
5					108.73 (99.25–124.14)

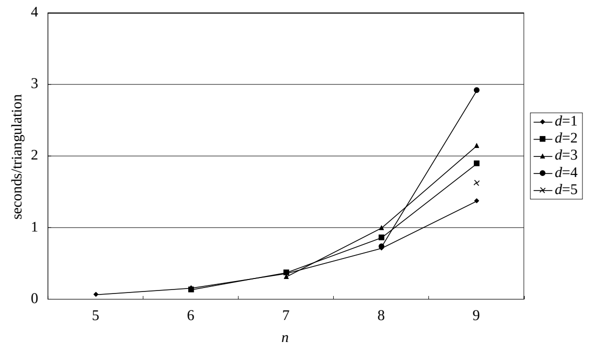

Fig. 1. Real computational time of enumeration in seconds per one triangulation. Averaged for twenty random configurations of n points in dimension d.

5. Enumeration of Spanning Regular Triangulations

We call a regular triangulation using all points *spanning*. We show that all spanning triangulations are connected by flips, and show their enumeration. The traces of the simplex method from vertices in the secondary polytope under the cost vector $\boldsymbol{w}_D$ with $w_{D\,i} = \|\boldsymbol{a}_i\| = \sum_{j=1}^{d}(a_{i,j})^2$ becomes the reverse search tree. We also consider the diameter of the secondary polytope using the arguments for this enumeration.

The first question concerning spanning regular triangulations is whether their corresponding vertices are connected by edges in the secondary polytope. To investigate this, consider the polytope obtained by lifting the points $\mathcal{A}$ by $\boldsymbol{w}_D$. By perturbing $\boldsymbol{w}_{\mathrm{D}}$, if necessary, we can assume that there are exactly $d+1$ points on any of the lower facets of this polytope. The corresponding regular triangulation is a Delaunay triangulation. We consider transforming a spanning regular triangulation into this Delaunay one.

Lemma 17 *From a spanning regular triangulation, we can generate a sequence of regular triangulations to one Delaunay triangulation by flips such that*
(1) *all the regular triangulations appearing in this process are spanning, and the*

inner product of $\boldsymbol{w}_{\mathrm{D}}$ *and the volume vector of a regular triangulation is strictly decreasing, and furthermore*
(2) *a circuit used in a flip in the sequence is never used again in this process.*

Proof. For each triangulation Δ, we consider a piecewise linear function $g_{\Delta}(\boldsymbol{x})$ on $\mathrm{conv}(\mathcal{A})$ such that $g_{\Delta}(\boldsymbol{a}_i) = w_{\mathrm{D}\,i}$ on points $\boldsymbol{a}_i$ used in the triangulation, and g_{Δ} linear on each d-simplex of the triangulation. Let $\Delta_{\boldsymbol{w}_{\mathrm{D}}}$ be the regular triangulation for the weight vector $\boldsymbol{w}_{\mathrm{D}}$. Then, by calculating the volume under piecewise linear functions $g_{\Delta_{\boldsymbol{w}_{\mathrm{D}}}}$, g_{Δ},

$$\int_{\mathrm{conv}(\mathcal{A})} g_{\Delta_{\boldsymbol{w}_{\mathrm{D}}}}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} < \int_{\mathrm{conv}(\mathcal{A})} g_{\Delta}(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$$

holds for any triangulation Δ except $\Delta_{\boldsymbol{w}_{\mathrm{D}}}$. Noting that this integral is for a piecewise linear function, the following holds

$$\langle \boldsymbol{w}_{\mathrm{D}}, \boldsymbol{\varphi}_{\Delta_{\boldsymbol{w}_{\mathrm{D}}}} \rangle = (d+1)\int_{\mathrm{conv}(\mathcal{A})} g_{\Delta_{\boldsymbol{w}_{\mathrm{D}}}}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} < (d+1)\int_{\mathrm{conv}(\mathcal{A})} g_{\Delta}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = \langle \boldsymbol{w}_{\mathrm{D}}, \boldsymbol{\varphi}_{\Delta} \rangle,$$

where $\langle \boldsymbol{w}, \boldsymbol{\varphi} \rangle$ is the inner product of $\boldsymbol{w}$ and $\boldsymbol{\varphi}$, and $\boldsymbol{\varphi}_{\Delta_{\boldsymbol{w}_{\mathrm{D}}}}$, $\boldsymbol{\varphi}_{\Delta}$ are the volume vectors of $\Delta_{\boldsymbol{w}_{\mathrm{D}}}$ and Δ.

Since for $\boldsymbol{w}_{\mathrm{D}}$ all the lifted points are on the boundary of their lower hull, for any triangulation, a flip which makes a point unused in any simplex necessarily increases the inner product of $\boldsymbol{w}_{\mathrm{D}}$ and the volume vector. Consider a linear programming problem of minimizing a linear function with $\boldsymbol{w}_{\mathrm{D}}$ as its cost vector on the secondary polytope. For a vertex corresponding to a non-Delaunay regular triangulation there exists an adjacent vertex connected by an edge whose inner product with $\boldsymbol{w}_{\mathrm{D}}$ strictly decreases. Hence, performing the corresponding flip, a new triangulation with smaller inner product value is obtained and this flip does not destroy the spanning property. Thus, (1) is shown.

For the sequence of triangulations $\Delta_0, \ldots, \Delta_k$ where Δ_k is the Delaunay triangulation, we see

$$g_{\Delta_i}(\boldsymbol{x}) \geq g_{\Delta_j}(\boldsymbol{x}) \quad (i < j;\ \boldsymbol{x} \in \mathrm{conv}(\mathcal{A})).$$

This is because for lifted points corresponding to a circuit Z their convex hull is a full-dimensional simplex in the lifted space and have upper and lower boundaries. Each of the upper and lower boundaries corresponds to a triangulation of Z in the original space. Since any circuit has two triangulations, these two are such ones, and hence strict above-below relation holds. If a circuit Z is used twice for flips for i and j with $i < j$, $g_{\Delta_i}(\boldsymbol{x}) = g_{\Delta_j}(\boldsymbol{x})$ for $\boldsymbol{x}$ in the interior of $\mathrm{conv}(Z)$, while by the argument above $g_{\Delta_i}(\boldsymbol{x}) > g_{\Delta_{i+1}}(\boldsymbol{x}) \geq g_{\Delta_j}(\boldsymbol{x})$, a contradiction. □

Theorem 18 *All the spanning regular triangulations can be enumerated in* $O(d^2 s^2\ \mathrm{LP}(n-d-1, s)\#\mathcal{R}_{\mathrm{spanning}})$ *time and* $O(ds)$ *working space, where* $\mathcal{R}_{\mathrm{spanning}}$ *is the set of spanning regular triangulations.*

Proof. We use similar arguments as for the enumeration of regular triangulations. The objects to be enumerated are $S = \mathcal{R}_{\text{spanning}}$ the spanning regular triangulations. When the i-th circuit is formed by a simplex and a point in its interior, we set $\mathrm{Adj}(T, i) = \emptyset$. Otherwise, $\mathrm{Adj}(T, i)$ is same as the case of regular triangulations. We introduce a total order on $\mathcal{R}_{\text{spanning}}$ by defining a triangulation with the inner product of its volume vector and $\boldsymbol{w}_{\mathrm{D}}$ smaller, to be larger. When there is a tie, we compare their volume vectors in lexicographic order. Then the claim holds similarly as Theorem 13: the conditions in Definition 5 are satisfied and the complexities are as in Proposition 16. The reverse search tree is the traces of the simplex method from vertices corresponding to spanning regular triangulations in the secondary polytope under the cost vector $\boldsymbol{w}_D$. □

The arguments in Lemma 17 can be further utilized as follows.

Theorem 19 *The diameter of the secondary polytope is* $O(n^{d+2})$.

Proof. Since the number of circuits is bounded by $O(n^{d+2})$, and the piecewise linear function monotonically changes downwards also for non-spanning regular triangulations. □

In Ref. [1], the authors construct the arrangement of $O(n^{d+2})$ hyperplanes whose cells correspond to the vertices of the secondary polytope, and two cells in the arrangement are adjacent each other if and only if the corresponding vertices of the secondary polytope are connected by an edge. From this, Theorem 19 can be obtained because any two cells in the arrangement are connected by a sequence of at most $O(n^{d+2})$ adjacent cells. However, the sequence from any regular triangulation to the Delaunay triangulation can be found by the arguments in Lemma 17 and Theorem 19.

6. Enumeration of Equivalence Classes of Regular Triangulations

In section 2, we showed an extension of reverse search for enumeration of equivalence classes of objects. In section 4.1, we gave a structure to enumerate regular triangulations. We combine these results and give an algorithm for the enumeration of equivalence classes of regular triangulations (subsection 6.1). The equivalence for the classes reflects the symmetry of the given point configuration. We also show applications to products of two simplices (subsection 6.2) and hypercubes (subsection 6.3).

6.1. *Enumerating Equivalence Classes of Regular Triangulations*

We define symmetries of point configurations by groups. A point configuration may be the set of vertices of a symmetric polytope, and the group a set of transformations which do not change the polytope.

Let G be some group of affine maps which define bijections on $\mathrm{conv}(\mathcal{A})$. These maps define bijections on the points $\mathcal{A}$. A bijection on $\mathrm{conv}(\mathcal{A})$ can be determined by its action on $\mathcal{A}$. So we can regard G as a subgroup of the symmetric group S_n

consisting of elements satisfying the conditions of affine bijectivity. We define an equivalence relation using this group.

Definition 20 (equivalence on simplices and triangulations) *Let $g \in G$.*

- *G acts on $\mathcal{A} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$.*
- *The action of G on a simplex of $\mathcal{A}$ is induced by the action on its vertices: $g \operatorname{conv}\{\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_m}\} = \operatorname{conv}\{g\boldsymbol{a}_{i_1}, \ldots, g\boldsymbol{a}_{i_m}\}$.*
- *The action of G on the triangulations of $\mathcal{A}$ is induced by the action on the simplices: $gT = \{g\sigma : \sigma \in T\}$.*
- *The action of G on the vertices, simplices or triangulations defines an equivalence relation on each of them: two elements are equivalent if they can move to each other by an element of G. These equivalences classifies these sets.*

Since G is a set of affine bijections, it maps a simplex to a simplex, and a triangulation to a triangulation. Since affine bijections only relabel the name of the vertices, for any $g \in G$ two triangulations T_1 and T_2 can be modified along a circuit if and only if gT_1 and gT_2 can. Thus, the definition of equivalence class on (regular) triangulations satisfy the last condition of Definition 7, and the conditions for a reverse search structure for equivalence classes with order are satisfied. Theorem 4 leads to the following.

Theorem 21 (enumerating equivalence classes of regular triangulations) *By the reverse search structure, total order and equivalence relation defined in Definition 11, 12 and 20, we can enumerate the equivalence classes of regular triangulations with respect to symmetry. The time complexity and required memory are as in Theorem 4. Time complexities for* time(Adj) *and* time(f) *are the same as the case of Theorem 13 and* time(rep) *in general is as in Lemma 22.*

Lemma 22 *The representative of an equivalence class is the maximum element. The time complexity* time(rep) *for finding the representative is $O(n\#G)$.*

Proof. Judging the order between two volume vectors can be done in n time. By checking all actions of G, we can obtain the above time complexity. □

6.2. *Products of Two Simplices*

We are interested in enumerating the triangulations for products of two simplices. We take as the standard d-simplex Δ_d the convex hull $\operatorname{conv}\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{d+1}\}$ in $\mathbb{R}^{d+1}$. We write $\boldsymbol{e}_i$ or $\boldsymbol{f}_j$ for unit vectors with i-th or j-th element one and the rest zeros. The product of two standard simplices $\Delta_k \times \Delta_l$ is

$$\Delta_k \times \Delta_l = \operatorname{conv}\left\{ \begin{pmatrix} \boldsymbol{e}_i \\ \boldsymbol{f}_j \end{pmatrix} \in \mathbb{R}^{k+l+2} : i \in \{1, \ldots, k+1\}, j \in \{1, \ldots, l+1\} \right\}.$$

Our objects to enumerate are the triangulations of $\mathcal{A} = \operatorname{vert}(\Delta_k \times \Delta_l)$, where $\operatorname{vert}(\Delta_k \times \Delta_l)$ are the vertices.

The product $\Delta_k \times \Delta_l$ has a symmetric structure: even if we commute the axes of each simplex, the shape of the product does not change.

Definition 23 *We formulate the symmetry of $\Delta_k \times \Delta_l$ by the action of the direct product of symmetric groups $S_{k+1} \times S_{l+1}$ to the vertices. An element $(g, h) \in S_{k+1} \times S_{l+1}$ acts on the vertices of $\Delta_k \times \Delta_l$ as $(g,h)\begin{pmatrix} \boldsymbol{e}_i \\ \boldsymbol{f}_j \end{pmatrix} = \begin{pmatrix} \boldsymbol{e}_{g(i)} \\ \boldsymbol{f}_{h(j)} \end{pmatrix}$.*

This action consists of affine maps defining bijections on $\Delta_k \times \Delta_l$ and $\mathrm{vert}(\Delta_k \times \Delta_l)$. Actions and equivalence relations on simplices and triangulations are induced as in Definition 20. As shown in Theorem 21, this equivalence relation satisfies the last condition of reverse search structure for classes with symmetry, and we can enumerate the classes of regular triangulations with respect to symmetry.

When $k = l$, there is further symmetry: commuting the first half of axes and the last half. This can be formulated as an action of $S_k \times S_k \times S_2$. We can also consider this action with some modifications on the arguments on complexity shown below.

The volume vectors can be regarded as matrices: $(\boldsymbol{\varphi}_T \begin{pmatrix} \boldsymbol{e}_i \\ \boldsymbol{f}_j \end{pmatrix})_{ij} \in \mathbb{R}^{k+1} \times \mathbb{R}^{l+1}$. $S_{k+1} \times S_{l+1}$ acts on a volume vector $\boldsymbol{\varphi}_T$ as rearrangements of rows and columns of a matrix. Two regular triangulations T and T' are in the same class if and only if their volume vectors $\boldsymbol{\varphi}_T$ and $\boldsymbol{\varphi}_{T'}$ are in the same class, since the correspondence between regular triangulations and volume vectors is one-to-one (cf. Theorem 9).

We introduced a total order on the regular triangulations by the lexicographic order of their corresponding volume vectors (Definition 12). For the case of $\Delta_k \times \Delta_l$, we can regard this order as a lexicographic order on matrices: a matrix (a_{ij}) is smaller than (b_{ij}) if for some (i_0, j_0), $a_{i_0 j_0} < b_{i_0 j_0}$, and for any (i, j) such that $i < i_0$ or such that $i = i_0$ and $j < j_0$, $a_{ij} = b_{ij}$.

As the representative of a class of regular triangulations we took the maximum one.

Lemma 24 *Given a regular triangulation T, the time* time(rep) *to compute the representative element of its equivalence class is $O(l!\, k^2 l^2)$.*

This is faster than the time complexity for the general case in Lemma 22.

Proposition 25 *The enumeration algorithm for equivalence classes of regular triangulations in Theorem 21, works for the case of $\Delta_k \times \Delta_l$. The time complexity is linear to the number of classes of regular triangulations, and the memory required is several times the size of a triangulation.*

6.3. *Hypercubes*

We are interested in enumerating the triangulations of hypercubes. We write $\boldsymbol{e}_i$ for unit vectors with the i-th element one and the rest zeros. The d-cube C_d (in $\mathbb{R}^{2d}$, different from the definition in section 3) is given by

$$C_d = \mathrm{conv}\left\{ \begin{pmatrix} \boldsymbol{e}_{i_1} \\ \vdots \\ \boldsymbol{e}_{i_d} \end{pmatrix} \in \mathbb{R}^{2d} : \boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_d} \in \mathbb{R}^2, i_1, \ldots, i_d \in \{1, 2\} \right\}.$$

Our objects to enumerate are the triangulations of $\mathcal{A} = \mathrm{vert}(C_d)$.

We define the symmetry of C_d as follows.

Definition 26 *We formulate the symmetry of C_d by an action of the direct product of $d+1$ symmetric groups $S_2 \times \cdots \times S_2 \times S_d$ to the vertices. An element $(g_1, \ldots, g_d, h) \in S_2 \times \cdots \times S_2 \times S_d$ acts on vertices of C_d as* $(g_1, \ldots, g_d, h) \begin{pmatrix} \boldsymbol{e}_{i_1} \\ \vdots \\ \boldsymbol{e}_{i_d} \end{pmatrix} = \begin{pmatrix} \boldsymbol{e}_{g_1(i_{h(1)})} \\ \vdots \\ \boldsymbol{e}_{g_d(i_{h(d)})} \end{pmatrix}$.

This action consists of affine maps defining bijections on C_d and $\mathrm{vert}(C_d)$. Actions and equivalence relations on simplices and triangulations are induced as in Definition 20. As shown in Theorem 21, this equivalence relation satisfies the last condition of reverse search structure for equivalence classes with symmetry, and we can enumerate the equivalence classes of regular triangulations with respect to symmetry.

Proposition 27 *The enumeration algorithm for equivalence classes of regular triangulations in Theorem 21, works for the case of C_d. The time complexity is linear to the number of classes of regular triangulations, and the memory required is several times the size of a triangulation.*

7. Enumeration of Maximal Independent Sets of a Graph

In section 8, we show that triangulations can be regarded as a subclass of the maximal independent sets of some graph. Efficient algorithms to enumerate maximal independent sets are known.[31,32] We reformulate one of these algorithms for our case.

Tsukiyama, Ide, Ariyoshi and Shirakawa proposed an algorithm to enumerate maximal independent sets of a graph.[32].This runs in time proportional to the number of maximal independent sets, but requires memory of the order of the size of the graph. Lawler, Lenstra and Rinnooy Kan extended this algorithm for the enumeration of maximal independent sets of independent set systems.[31] The time complexity is proportional to the number of maximal independent sets, with evaluation based on the number of executions of independent tests. The algorithm requires memory of the size of the base set, which is equal to the number of the vertices in the case of a graph. For our case, the vertices of the graph correspond to all of the d-simplices, which can become large.

We reformulate algorithm Ref. [31] to the graph case. Our algorithm does not put the graph itself in the memory, but proceeds by testing whether two vertices are connected by an edge. Also, we reduce the time complexity compared to just applying Ref. [31] to graphs.

Let the base set be $E = \{1, \ldots, n\}$, and $\mathcal{M}$ the set of maximal independent sets. Let us denote by $\mathcal{M}_j$ the family of independent sets that are maximal within $\{1, \ldots, j\}$. In this algorithm, $\mathcal{M}_j$ is computed using $\mathcal{M}_{j-1}$, starting from $\mathcal{M}_0 = \{\emptyset\}$, to obtain $\mathcal{M}_n = \mathcal{M}$.

The update from $\mathcal{M}_{j-1}$ to $\mathcal{M}_j$ is done as follows. For each I in $\mathcal{M}_{j-1}$, the independence of $I \cup \{j\}$ is tested. If this is independent, $I \cup \{j\}$ is added to $\mathcal{M}_j$. If not independent, I and other maximal independent sets of $\mathcal{M}_j$ included in $I \cup \{j\}$ become candidates to be added. If I' is such maximal independent set of $\mathcal{M}_j$ included in $I \cup \{j\}$, it should be maximal in $I \cup \{j\}$. This fact is used in reverse: first the maximal independent sets in $I \cup \{j\}$ are listed, and then their maximal independence in $\mathcal{M}_j$ is checked. The algorithm elaborates to produce I' from a single I. The computation can be regarded as a search on a tree. The tree is rooted by the $\emptyset$, and nodes at level j correspond to members of $\mathcal{M}_j$. For each I in $\mathcal{M}_{j-1}$, the corresponding I' (possibly several) in $\mathcal{M}_j$ become its children. The maximal independent sets, the leaves of the tree, are enumerated by depth first search.

Our formulation is for the enumeration of maximal independent sets of a simple undirected graph. The base set E is the set of vertices of the graph. We suppose the existence of an oracle which answers in unit time the previous or next vertex in the adjacency list for a given vertex. This seems trivial when we write $E = \{1, \ldots, n\}$, but it is not for our case, because the vertices correspond to the d-simplices $\mathcal{S}$. The existence of such an oracle is discussed in section 8.3.

Let $m = \max_{I \in \mathcal{M}} \#I$ be the maximum cardinality of vertices in a maximal independent set. We say that two vertices are *intersecting* if they are connected by an edge, and denote by time(intersect) the time needed to judge whether two vertices are intersecting or not.

Algorithm 28 (enumerating maximal independent sets of a graph)
Step 1. *For each I in $\mathcal{M}_{j-1}$ we want to find all independent sets I' maximal within $I \cup \{j\}$. For this, we only have to check the intersection of the newly added vertex j with the no more than m current ones in I.*

(1) *If j does not intersect any vertex in I, $I \cup \{j\}$ is the only maximal independent set in $I \cup \{j\}$. Furthermore, this is maximal independent in $\{1, \ldots, j\}$.*

(2) *If j intersects some of the vertices in I,*

   (a) *I is a maximal set in $I \cup \{j\}$. Furthermore, I is maximal independent in $\{1, \ldots, j\}$.*

   (b) *$I' = \{i \in I \cup \{j\} : i$ does not intersect $j\}$ is the other maximal independent set of $I \cup \{j\}$. We check the maximality of I' in $\{1, \ldots, j\}$ by testing if some $i \in \{1, \ldots, j\} \setminus I'$ does not intersect any vertex in I'. If such i exists, I' is not maximal independent, and if not, it is maximal independent.*

Step 2. *We check whether I' is obtained from the lexicographically smallest $I \in \mathcal{M}_{j-1}$. This is always true for cases* (1) *and* (2a). *We only have to check for the case* (2b).

We have to check for each $i < j$, $i \notin I$, the independence of $(I' \setminus \{j\}) \cup (I \cap \{1, \ldots, i-1\}) \cup \{i\}$. Each of these independence tests can be done by checking *whether i intersects some vertex in $(I' \setminus \{j\}) \cup (I \cap \{1, \ldots, i-1\})$. If i intersects some vertex, $(I' \setminus \{j\}) \cup (I \cap \{1, \ldots, i-1\}) \cup \{i\}$ is not independent. If i does not intersect any of the vertices, this set is independent. For this latter case, I' can be obtained from another lexicographically smaller I, and I' is rejected. If this does not happen for any $i < j$, $i \notin I$, I' is the child of I, and should be retained.*

The time complexity is as follows.

In **Step 1**, for each I in $\mathcal{M}_{j-1}$ the candidates I' we take are one or two. The total time complexity for this step is $O(m \cdot \text{time(intersect)} n \# \mathcal{M})$. For the extra time complexity for the case (2a), each independence test can be done in $m \cdot \text{time(intersect)}$ time, and it takes $m \cdot \text{time(intersect)} n$ time for each I'. The total time complexity for this step is $O(m \cdot \text{time(intersect)} n^2 \# \mathcal{M})$.

In **Step 2**, the independence test can be done in time $m \cdot \text{time(intersect)}$. The total time complexity for this step is $O(m \cdot \text{time(intersect)} n^2 \# \mathcal{M})$.

The time complexity analyzed above is the total time needed for descending the search tree. Since we want to restrict the required memory size to $2m$, we have to recompute the parent when ascending the tree. However, this does not increase the order of the time complexity.

Suppose we are at $I' \in \mathcal{M}_{j+1}$ and want to find its parent $I \in \mathcal{M}_j$. If $j+1 \notin I'$, I' is the case (2a), and $I = I'$. When $j+1 \in I'$, we try to add $\max(I' \setminus \{j+1\})+1, \ldots, j$ in this order to obtain I. Recall that I was lexicographically the smallest among the possible parents. If we have no element to add, I' is the child for case (1) in **Step 1**, and if we have, I' is the the case (2a).

The time complexity needed for a recomputation of I is $m \cdot \text{time(intersect)} n$ and $O(m \cdot \text{time(intersect)} n^2 \# \mathcal{M})$ as a whole. Thus ascending the tree does not increase the order of the time complexity.

Next, we discuss the space complexity.

To identify which node we are, the information of the current and previous independent sets and the depth j is enough. This requires memory of size $2m$ We do not need to store the path from the root to the current node.[31] This is realized as follows.

If we are descending the search tree, the next thing to do is to compute the candidates I' and descend the tree. For case (1), we descend to the only candidate $I' = I \cup \{j\}$. For case (2), let us descend first to the candidate $I' = I$. We try the second candidate $\{i \in I \cup \{j\} : \text{ not intersecting } j\}$, if it is a child, when we come back to this node ascending from $I \in \mathcal{M}_j$.

Since we have the oracle mentioned above, we do not have to store all vertices. Thus, we can traverse the search tree only with the information of our current and previous independent sets and the depth j.

Theorem 29 *Algorithm 28 enumerates all maximal independent sets of a simple undirected graph. This works in time $O(m \cdot \text{time(intersect)} n^2 \# \mathcal{M})$ with memory size $2m$. The graph has n vertices, with the maximum cardinality of independent sets m, and* time(intersect) *is the time required to compute if two vertices are connected* *by and edge.*

The computation of this enumeration can be divided into smaller problems, and performed in parallel. This can be done by enumerating all nodes of depth not larger than j, which are $\mathcal{M}_j$, the maximal independent sets of $\{1, \ldots, j\}$, and performing searches for each subtrees with roots $I \in \mathcal{M}_j$.

8. Enumeration of Triangulations

We propose an algorithm to enumerate all triangulations, regular or not, for arbitrary configurations of points in general dimension. We formulate triangulations as maximal independent sets of a graph, and apply the maximal independent set enumeration algorithm proposed in section 7. The graph here is the graph with all maximal dimensional simplices as the vertices and edges between those simplices intersecting improperly. This algorithm works in time proportional to the number of maximal independent sets. The memory required is twice the size of a maximal independent set. We also show an application of this algorithm to the case of polytopes of the products of two simplices.

We first define the intersection graph (subsection 1), and enumerate triangulations as maximal independent sets of this graph (subsection 2). We discuss further two basic operations used in the enumeration: the enumeration of maximal dimensional simplices (subsection 3) and testing whether two simplices are intersecting improperly or not (subsection 4). We also show the enumeration for products of two simplices in which some parts of the algorithm can be made faster (subsection 5).

8.1. *Triangulations as Maximal Independent Sets*

See section 3 for definitions of triangulations.

Definition 30 • **(intersection graph)** *The* intersection graph *of $\mathcal{S}$ is the graph with $\mathcal{S}$ the vertices and edges between two simplices intersecting improperly.*

- $\mathcal{I} = \{I \in 2^{\mathcal{S}} :$ *independent set of the intersection graph of* $\mathcal{S}\}$
- $\mathcal{M} = \{I \in 2^{\mathcal{S}} :$ *maximal independent set of the intersection graph of* $\mathcal{S}\}$
- $\mathcal{T} = \{I \in 2^{\mathcal{S}} :$ *triangulation of* $\mathcal{A}\}$

Trivially, $\mathcal{M}$ is a subclass of $\mathcal{I}$. Let I be a triangulation. The d-simplices in I must not intersect improperly, so I is an independent set. Furthermore, since we cannot add anymore d-simplex to a triangulation without making improper intersections, I is an maximal independent set. This gives the following proposition.

Proposition 31 *$\mathcal{T}$ is a subclass of $\mathcal{M}$. An element $I \in \mathcal{M}$ is in $\mathcal{T}$ if and only if the sum of the volume of the d-simplices in I is equal to the volume of* $\mathrm{conv}(\mathcal{A})$.

In the next section we enumerate the triangulations $\mathcal{T}$ by giving an algorithm to enumerate the maximal independent sets $\mathcal{M}$.

The difference of $\mathcal{T}$ and $\mathcal{M}$ becomes a loss. This kind of thing happens, for example, for the point configuration given as the vertices of Schönhardt's polyhedron (cf. 10.2.1 of Ref. [26], Ref. [33]). This polyhedron is a concave polyhedron made by twisting a little bit a triangle of a prism. No tetrahedron with vertices among the six vertices is included in this polyhedron. The set made by the three tetrahedra fitting the outer concave part of this polyhedron becomes a maximal independent set of the convex polytope of the six vertices. However, this is not a triangulation, because the inner part is left. Whether this kind of thing happens or not depends on the point configuration, though this dependence is not easy to detect.

8.2. *Enumerating Triangulations*

Now we apply the formulation above to the enumeration of triangulations. The base set E is the set of d-simplices $\mathcal{S}$. We suppose the existence of an oracle which gives in unit time the previous or next simplex for a given simplex for some fixed order of E. The existence of such oracle is discussed in subsection 8.3. The number $m = \max_{I \in \mathcal{M}} \#I$ is the maximum cardinality of simplices in a maximal independent set, and time(intersect) is the time needed to judge whether two simplices are intersecting properly or not.

Theorem 32 (enumerating triangulations) *By Algorithm 28, we can enumerate all maximal independent sets, thus the triangulations, of the intersection graph of* $\mathcal{S}$. *This works in* $O(m \cdot \mathrm{time(intersect)}(\#\mathcal{S})^2 \#\mathcal{M})$ *time with memory size* $2m$.

The number of simplices in a triangulation is bounded by m. If m, the largest cardinality of (maximal) independent sets, and the largest cardinality of a triangulation is the same, the required memory becomes only twice the size of a triangulation.

8.3. *Enumerating d-simplices*

We suppose the existence of an oracle which gives in unit time the previous or next d-simplex for a given d-simplex for some fixed order of the d-simplices $\mathcal{S}$.

The given point configuration is $\mathcal{A} = \{\boldsymbol{a}_1, \dots, \boldsymbol{a}_n\}$. A set of $d+1$ points $\{\boldsymbol{a}_{i_1}, \dots, \boldsymbol{a}_{i_{d+1}}\}$ becomes the set of vertices for a d-simplex if and only if $\binom{\boldsymbol{a}_{i_1}}{1}, \dots, \binom{\boldsymbol{a}_{i_{d+1}}}{1}$ are linearly independent.

Thus the problem reduces to the existence of a similar oracle for the bases of $\{\binom{\boldsymbol{a}_1}{1}, \dots, \binom{\boldsymbol{a}_n}{1}\}$. This can be realized by reverse search with the time complexity $O((d+1)n\#\mathcal{S})$ for the whole enumeration.[14] For a given base, answering its next or previous base can be done in time approximately $O((d+1)n)$.

Using this oracle, we do not need to store all of the d-simplices, but memory for only several times the size of a simplex is enough. This enables handling of large problems. For problems that even the memory for simplices matters, the enumeration of all triangulations might require hopelessly enormous time. However, since the algorithm does work, we can at least perform partial enumeration.

For smaller problems for which we can store all of the d-simplices, it is better to enumerate and store them. This is much faster than asking the oracle each time. The enumeration here can be done by the reverse search as mentioned above. Though in practice, trying all $d+1$ points among S and checking if it is a d-simplex by calculating the determinant of the corresponding $d+1$ vectors is fast enough.

8.4. *Testing the Intersection of d-simplices*

Computing whether two simplices are intersecting improperly or not is usually the most time consuming calculation. The time complexity in Theorem 32 was dominated by the number of this calculation. Here, we give algorithms and their complexity for this calculation. (A matrix is regarded as a set of column vectors.)

Algorithm 33 (testing the intersection of d-simplices)
Input: $\{\boldsymbol{p}_1, \dots, \boldsymbol{p}_{d+1}\}$, $\{\boldsymbol{q}_1, \dots, \boldsymbol{q}_{d+1}\}$ *: vertices of d-simplices in* $\mathbb{R}^d$
Output: *whether the simplices are intersecting improperly or not*
Suppose $\{\boldsymbol{p}_1, \dots, \boldsymbol{p}_{d+1}\} \cap \{\boldsymbol{q}_1, \dots, \boldsymbol{q}_{d+1}\} = \emptyset$. *First, by affine transformation, we move* $(\boldsymbol{q}_1 \cdots \boldsymbol{q}_{d+1})$ *to* $(\boldsymbol{0}\ \boldsymbol{e}_1 \cdots \boldsymbol{e}_d)$, *where* $\boldsymbol{e}_i$ *are the unit vectors. The points* $(\boldsymbol{p}_1 \cdots \boldsymbol{p}_{d+1})$ *move to* $(\boldsymbol{q}_2 - \boldsymbol{q}_1 \cdots \boldsymbol{q}_{d+1} - \boldsymbol{q}_1)^{-1}(\boldsymbol{p}_1 - \boldsymbol{q}_1 \cdots \boldsymbol{p}_{d+1} - \boldsymbol{q}_1)$. *Let C denote this matrix. The convex hull of these points has a point common with the convex hull* $\mathrm{conv}\{\boldsymbol{0}\ , \boldsymbol{e}_1, \dots, \boldsymbol{e}_d\}$ *if and only if these simplices are intersecting improperly. This is equivalent to whether inequalities*

$$\begin{pmatrix} C \\ 1 \cdots 1 \\ -1 \cdots -1 \\ -\sum_i C_{i\cdot} \end{pmatrix} \boldsymbol{x} \geq \begin{pmatrix} \boldsymbol{0} \\ 1 \\ -1 \\ -1 \end{pmatrix},$$

$$\boldsymbol{x} \geq \boldsymbol{0}\ ,$$

where a solution $C\boldsymbol{x}$ becomes a point in common mentioned above, have a feasible solution or not. When $\{\boldsymbol{p}_1, \dots, \boldsymbol{p}_{d+1}\}$ *and* $\{\boldsymbol{q}_1, \dots, \boldsymbol{q}_{d+1}\}$ *have points in common, the testing reduces to smaller linear program, after neglecting by projection the dimensions spanned by the points in common.*

Lemma 34 *Algorithm 33 works in time* $\mathrm{LP}(d+1, d+3)$, *where* $\mathrm{LP}(n, m)$ *is the time required to solve a linear programming problem with m constraints and n variables.*

If it is possible to store if the simplices are intersecting properly or improperly for all pairs of simplices, it is better to compute first all the intersections and store them. This requires memory of $\binom{\#\mathcal{S}}{2}$ bits. It can be done in time $\mathrm{time(intersect)}\binom{\#\mathcal{S}}{2}$. Since this computation is just to test intersection for $\binom{\#\mathcal{S}}{2}$ pairs, it can obviously be divided and computed in parallel. By this preprocessing we can remove the factor time(intersect) of $\mathcal{M}$ from the time complexity in Theorem 32 to achieve $O(\mathrm{time(intersect)}(\#\mathcal{S})^2 + m(\#\mathcal{S})^2\#\mathcal{M})$.

8.5. *Products of Two Simplices*

We are interested in enumerating the triangulations for products of two simplices. The definition of this polytope was given in subsection 6.2.

First we state several lemmas for later use. The volume of $(k+l)$-simplices in a triangulation of $\Delta_k \times \Delta_l$ is constant. Under scaling, they have volume $1/(k+l)!$, and the product has volume $1/k!l!$. This leads to the following.

Lemma 35 *No more than $m = (k+l)!/k!l!$ $(k+l)$-simplices are included in an independent set of the intersection graph of $\Delta_k \times \Delta_l$. All triangulation consists of m $(k+l)$-simplices.*

The $(k+l)$-simplices in $\Delta_k \times \Delta_l$ correspond to the spanning trees of the complete bipartite graph $K_{k+1,l+1}$ (7.3.D. of Ref. [2]). This proves the next lemma.

Lemma 36 *The number of $(k+l)$-simplices of $\Delta_k \times \Delta_l$ is $(k+1)^l(l+1)^k$.*

The generation of spanning trees of $K_{k+1,l+1}$ can be done using a constant time per tree with small memory.[34,35] Thus we can generate the corresponding $(k+l)$-simplices similarly.

Lemma 37 (enumerating d-simplices: the $\Delta_k \times \Delta_l$ case) *We can generate the $(k+l)$-simplices of $\Delta_k \times \Delta_l$ using a constant time per simplex with small memory. Thus, the oracle required for Algorithm 28 exists.*

For the point configuration of $\Delta_k \times \Delta_l$, testing whether two simplices are intersecting improperly or not can be reduced to judging the existence of a cycle in a subgraph of a directed $K_{k+1,l+1}$ (Lemma 2.3. of Ref. [20]), which leads to the time complexity. The intersection test for this $\Delta_k \times \Delta_l$ case can be computed faster using this graph property than by Algorithm 33.

Lemma 38 (testing the intersection of d-simplices: the $\Delta_k \times \Delta_l$ case) *Given two $(k+l)$-simplices in $\Delta_k \times \Delta_l$, judging whether they are intersecting improperly or not can be done in time $O(k+l)$.*

We apply Theorem 32 to the case of $\Delta_k \times \Delta_l$.

Theorem 39 (enumerating triangulations for $\Delta_k \times \Delta_l$) *For the point configuration $\mathcal{A} = \mathrm{vert}(\Delta_k \times \Delta_l)$, Algorithm 28 enumerates all maximal independent sets of the intersection graph of $\mathcal{S}$, thus the triangulations, in $O(\binom{k+l}{k}(k+l)k^{2l}l^{2k}\#\mathcal{M})$ time with memory size $2\binom{k+l}{k}$.*

Proof. By Lemma 37, the oracle exists. By Lemma 35, and $m = \binom{k+l}{k}$. By Lemma 36, $\#\mathcal{S} = (k+1)^l(l+1)^k$. By Lemma 38, time(intersect) $= O(k+l)$. □

8.6. *Enumerating Dissections*

By changing the intersection graph in Definition 30 to a graph with edges between d-simplices whose interior intersect, we can make Algorithm 28 enumerate dissections instead of triangulations. A *dissection* of a point configuration uses d-simplices with vertices among the given points as well, but instead of d-simplices intersecting at their faces, only requires them not to share interior points. To the authors knowledge, this is the only reasonable algorithm for enumerating dissections. It was used in computing dissections in Refs. [36,37].

Acknowledgments

We would like to thank Nobuki Takayama of Kobe University for introducing to us these problems and giving us lectures on many mathematical aspects of them. Comments from the referees improved the writing of this paper. The work of H.I. was supported in part by a Grant-in-Aid of the Ministry of Education, Japan.

Hiroshi Imai: `imai@is.s.u-tokyo.ac.jp`
Keiko Imai: `imai@ise.chuo-u.ac.jp`
Fumihiko Takeuchi: `fumi@is.s.u-tokyo.ac.jp`

References

1. L. J. Billera, P. Filliman, and B. Sturmfels, Constructions and complexity of secondary polytopes, *Advances in Math.* **83** (1990) 155–179.
2. I. M. Gelfand, M. M. Kapranov, and A. V. Zelevinsky, Discriminants, resultants and multidimensional determinants, Birkhäuser, Boston 1994.
3. I. M. Gel'fand, A. V. Zelevinskiĭ, and M. M. Kapranov, Newton polyhedra of principal $\mathcal{A}$-determinants, *Soviet Math. Dokl.* **40** (1990) 278–281.
4. J. A. de Loera, B. Sturmfels, and R. R. Thomas, Gröbner bases and triangulations of the second hypersimplex, *Combinatorica* **15** (1995) 409–424.
5. C. W. Lee, Regular triangulations of convex polytopes, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 4, Amer. Math. Soc. 1991, 443–456.
6. B. Sturmfels, Gröbner bases of toric varieties, *Tôhoku Math. J.* **43** (1991) 249–261.
7. B. Sturmfels, Gröbner bases and convex polytopes, American Mathematical Society, 1996.
8. B. Joe, Three-dimensional triangulations from local transformations, *SIAM J. Sci. Stat. Comput.* **10** (1989) 718–741.
9. B. Joe, Construction of three-dimensional Delaunay triangulations using local transformations, *Computer Aided Geometric Design* **8** (1991) 123–142.
10. H. Edelsbrunner and N. R. Shah, Incremental topological flipping works for regular triangulations, *Algorithmica* **15** (1996) 223–241.
11. M. A. Facello, Implementation of a randomized algorithm for Delaunay and regular triangulations in three dimensions, *Computer-Aided Geometric Design* **12** (1995) 349–370.
12. J. A. de Loera, Computing triangulations of point configurations, Preprint, December 1994.
13. D. Avis and K. Fukuda, A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra, *Discrete Comput. Geom.* **8** (1992) 295–313.
14. D. Avis and K. Fukuda, Reverse search for enumeration, *Discrete Appl. Math.* **65** (1996) 21–46.
15. H. Edelsbrunner, Algorithms in combinatorial geometry, Springer-Verlag, 1987.
16. T. Masada, An output size sensitive algorithm for the enumeration of regular triangulations, Technical Report 94-1, Department of Information Science, University of Tokyo, November 1994.
17. T. Masada, An algorithm for the enumeration of regular triangulations, Master's Thesis, Department of Information Science, University of Tokyo, 1995.

18. T. Masada, H. Imai, and K. Imai, Enumeration of regular triangulations, *Proc. 12th Annual ACM Symposium on Computational Geometry*, New York 1996, pp. 224–233.
19. `http://www-imai.is.s.u-tokyo.ac.jp/~fumi`
20. J. A. de Loera, Nonregular triangulations of products of simplices, *Discrete Comput. Geom.* **15** (1996) 253–264.
21. B. McKay, Isomorph-free exhaustive generation, *J. of Algorithms*, **26** (1998) 306–324.
22. S. Bespamyatnikh, An efficient algorithm for enumeration of triangulations *Comput. Geom.: Theory and Appl.*, to appear.
23. J. A. de Loera, S. Hoşten, F. Santos, and B. Sturmfels, The polytope of all triangulations of a point configuration, *Doc. Math.* **1** (1996) 103–119.
24. J. Rambau, TOPCOM—Triangulations Of Point Configurations and Oriented Matroids, `http://www.zib.de/rambau/TOPCOM/`
25. F. Takeuchi and H. Imai, Enumerating triangulations for products of two simplices and for arbitrary configurations of points, *Proc. Computing and combinatorics: 3rd annual international conference*, Lecture Notes in Computer Science 1276, Springer-Verlag, Berlin, pp. 470–481.
26. J. O'Rourke, Art gallery theorems and algorithms, Oxford University Press, New York, 1987.
27. G. M. Ziegler, Lectures on polytopes, Springer-Verlag, New York, 1995.
28. R. P. Stanley, The number of faces of a simplicial convex polytope, *Adv. Math.* **35** (1980) 236–238.
29. B. Chazelle, An optimal convex hull algorithm and new results on cuttings, *Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science*, 1991, pp. 29–38.
30. R. Seidel, Constructing higher-dimensional convex hulls at logarithmic cost per face, *Proc. 18th Annual ACM Symposium on the Theory of Computing*, 1986, pp. 404–413.
31. E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, Generating all maximal independent sets: NP-Hardness and polynomial-time algorithms, *SIAM J. Comput.* **9** (1980) 558–565.
32. S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, A new algorithm for generating all the maximal independent sets, *SIAM J. Comput.* **6** (1977) 505–517.
33. E. Schönhardt, Über die Zerlegung von Dreieckspolyedern in Tetraeder, *Math. Ann.* **98** (1928) 309–312.
34. S. Kapoor and H. Ramesh, Algorithms for enumerating all spanning trees of undirected and weighted graphs, *SIAM J. Comput.* **24** (1995) 247–265.
35. A. Shioura, A. Tamura, and T. Uno, An optimal algorithm for scanning all spanning trees of undirected graphs, *SIAM J. Comp.* **26** (1997) 678–692.
36. A. Below, U. Brehm, J. A. de Loera, J. Richter-Gebert, Minimal dissections and triangulations of 3-polytopes, *Discrete Comput. Geom.* **24** (2000) 35-48.
37. J. A. de Loera, F. Takeuchi, F. Santos, Extremal properties for dissections of convex 3-polytopes, *SIAM J. Discrete Math.* **14** (2001) 143-161.