Enumerating Triangulations in General Dimension

Fumihiko Takeuchi Tomonari Masada Hiroshi Imai
Department of Information Science, University of Tokyo
Hongo, Bunkyo-ku, Tokyo, 113-0033 Japan

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

April 30, 1998

Abstract

We propose algorithms to enumerate (1) regular triangulations, (2) spanning regular triangulations, (3) classes of regular triangulations in respect of 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. To accomplish these efficiency, we use reverse search technique by Avis and Fukuda, its extension for enumerating classes of objects, and a reformulation of a maximal independent set enumeration algorithm.

1 Introduction

Triangulations have been one of the main topics in computational geometry and other fields in recent years. Especially, some types of triangulations are found to bridge geometric issues and algebraic ones. Regular triangulations are of such a type [3, 12, 13]. For example, this subclass of triangulations has a close connection with a well-known paradigm of computer algebra, Gröbner bases, and also with theory of discriminants, hypergeometric functions, etc. (see [3, 8, 12, 19, 29, 30]). 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 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 case. 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, Delaunay triangulation cannot always be obtained from a non-regular triangulation by Delaunay flips [15, 16]. However, restricting ourselves to the class of regular triangulations, such a result is already shown in any dimensions [3, 12, 13]. Also, there are several works in computational geometry on regular triangulations such as [10, 11].

(1) Enumeration of regular triangulations. Enumeration of all regular triangulations is interesting from the viewpoint of computer-aided mathematical research. As mentioned above, regular triangulations have connection with many mathematical concepts, and by enumerating them mathematical problems can be investigated through computational experiments (e.g., see

[5, 8, 30]). Also, for the three-dimensional case, through the enumeration algorithm, exhaustive and local search can be performed for triangulations of three-dimensional objects in computer graphics, finite element method, etc.

By extending the original work by Masada [20, 21], we first propose an output-size sensitive and work-space efficient algorithm for enumerating regular triangulations of n points in the d-dimensional space. To achieve these efficiency, we used reverse search technique [1, 2]. The algorithm makes full use of the existing results on the secondary polytope that the vertices correspond to regular triangulations. These 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 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 working space is quite small.

There have been proposed three algorithms for enumerating regular triangulations [3, 5, 20, 21].

1. The algorithm by Billera, Filliman and Sturmfels [3] 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 [9] 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 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 this line.
2. An output-size sensitive algorithm is given by De Loera [5]. It is based on the breadth-first search enumeration, and is implemented using `Maple`. Since it is based on the breadth-first search, its work-space complexity is $\Omega(\#\mathcal{R})$, which becomes prohibitively large even for small-size problems.
3. An output-size sensitive and work-space efficient algorithm is originally developed by Masada [20, 21]. It is based on the reverse search technique developed in [1, 2], and is implemented in C.

The algorithm presented in this paper is a refined version of the last one, together with some new results for spanning triangulations, and have theoretical merits as described above. Practical merits of this algorithm will be seen from subsection 4.6. The codes are available via internet [34]. 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 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 classes of regular triangulations. As mentioned above, regular triangulations have connection with 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 enumerated the triangulations, all of which are regular, for the case of $\Delta_2 \times \Delta_3$ and $\Delta_2 \times \Delta_4$ [5, 6]. 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 the memory, and the memory constraint becomes serious in larger cases.

We propose an algorithm to enumerate classes of objects by reverse search. And then apply this to the enumeration of classes of regular triangulations in respect of symmetry for 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 of a triangulation.

(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 $\Delta_3 \times \Delta_3$. So, it is important also to enumerate all triangulations, regular or not. Though there are some results [7], there is no efficient algorithm to enumerate all triangulations in dimension higher than two. Our algorithm 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 graph case, 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. [23, 10.2.1]) for example. However we are thinking that the gap may be small even in higher dimension. 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, the vertices of the intersection graph, increases exponential to the dimension, but we cope with this by using their correspondence with spanning trees of an bipartite graph, and memorizing one simplex, or spanning tree, at once.

An implementation of this algorithm in *Mathematica* is available via internet [34].

Outline of this paper. We begin by a brief explanation of reverse search, and then give our formulation of reverse search for 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 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.

2 Reverse search

Avis and Fukuda introduced an enumeration technique called reverse search [2]. 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 algorithm (subsection 2.1), and then

show our extension for enumeration of 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 needed output-size sensitive memory to memorize all reached objects. To save memory, in addition to the adjacency relation, which is necessary for BFS and DFS, parent-children relation is needful for reverse search [1, 2].

First we state the adjacency and parent-children relation for reverse search. This structure for reverse search is named "local search structure given by an A-oracle." We call it a *reverse search structure* here.

Definition 2.1 (reverse search structure [2])

$(S, \delta, \mathrm{Adj}, f)$ *is a* reverse search structure *if it suffices 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 founded by applying f several times to an object.

Algorithm 2.2 (reverse search [2])

ReverseSearch$(\delta, \mathrm{Adj}, f, r)$
$v := r \quad j := 0$
repeat
  while $j < \delta$ **do**
    $j := j + 1 \quad next = \mathrm{Adj}(v, j)$
    if $next \neq \emptyset$ **then**
      if $f(next) = v$ **then**
        $\{v := next \quad j := 0\}$
  if $v \neq r$ **then**
    $u := v \quad v := f(v)$
    $j := 0$
    repeat $j := j + 1$
    until $\mathrm{Adj}(v, j) = u$
until $v = r$ *and* $j = \delta$

Theorem 2.3 ([2, Corollary 2.3.])

Algorithm 2.2 works for the reverse search structure in Definition 2.1. 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 classes

Later, we will give an algorithm to enumerate classes of regular triangulations. This will be based on the enumeration of 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 $\mathrm{rep}(f(a))$.

Definition 2.4 (reverse search structure for classes)

$(S, \delta, \mathrm{Adj}, f, \sim, \mathrm{rep})$ *is a* reverse search structure for 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 2.5

For the reverse search structure for classes in Definition 2.4, we can enumerate the 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 2.1.*

- $S/\sim = \{[a] : a \in S\}$ *is the set we want to enumerate*
- δ *is the same as the original reverse search structure*
- $\mathrm{Adj}([a], i) = \begin{cases} [\mathrm{Adj}(\mathrm{rep}(a), i)] & \text{if } \mathrm{Adj}(\mathrm{rep}(a), i) \neq \emptyset \text{ and } [\mathrm{Adj}(\mathrm{rep}(a), i)] \neq [\mathrm{rep}(a)] \text{ and} \\ & \text{if } [\mathrm{Adj}(\mathrm{rep}(a), i)] \neq [\mathrm{Adj}(\mathrm{rep}(a), j)] \text{ for any } j < i \\ \emptyset & \text{otherwise} \end{cases}$
- $f([a]) = [f(\mathrm{rep}(a))]$

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.*

Proof. We have to check conditions (1) to (5) of Definition 2.1 and the symmetry of the adjacency relation.
Symmetry of the adjacency: if $\mathrm{Adj}([\mathrm{rep}(a)], i) = [\mathrm{rep}(b)]$, $\mathrm{Adj}(\mathrm{rep}(a), i) = c$ for some $c \sim \mathrm{rep}(b)$, and $[\mathrm{rep}(a)] \neq [c]$. By $\mathrm{rep}(a)$ adjacent to c and $c \sim \mathrm{rep}(b)$, there exists $d \sim \mathrm{rep}(a)$ adjacent to $\mathrm{rep}(b)$. Thus $\mathrm{Adj}(\mathrm{rep}(b), j) = d$ for some j, and $[\mathrm{rep}(b)] = [c] \neq [\mathrm{rep}(a)] = [d]$. This implies $\mathrm{Adj}([\mathrm{rep}(b)], k) = [d] = [\mathrm{rep}(a)]$ for some k.
(1). (2) and (3) are satisfied by definition.
(4): if $f([a]) = [b]$ and $[a] \neq [b]$, $f(\mathrm{rep}(a)) \sim b \not\sim a$. Thus $f(\mathrm{rep}(a)) \not\sim \mathrm{rep}(a)$. Since $f(\mathrm{rep}(a)) \neq \mathrm{rep}(a)$, $\mathrm{Adj}(\mathrm{rep}(a), i) = f(\mathrm{rep}(a))$ for some i. Thus $\mathrm{Adj}([\mathrm{rep}(a)], j) = [f(\mathrm{rep}(a))]$ for some j. This leads $\mathrm{Adj}([a], j) = [b]$.
First statement of condition (5): we prove that the only class $[a]$ with $f([a]) = [a]$ is the class

which includes the root object r of the original reverse search structure. Since $r = \mathrm{rep}(f(r))$, r is the representative of its class $[r]$. For this class $f([r]) = [f(r)] = [r]$. If $[a]$ is a class where $f([a]) = [a]$, we have $[f(\mathrm{rep}(a))] = [a]$, thus $\mathrm{rep}(f(\mathrm{rep}(a))) = \mathrm{rep}(a)$, which implies $r = \mathrm{rep}(a) \in [a]$.
Second statement of condition (5): for any class $[a] \neq [r]$, $\mathrm{rep}(a) \neq r$, and there exists $n \in \mathbb{N}$ such that $(\mathrm{rep} \circ f)^{(n)}(\mathrm{rep}(a)) = r$. Thus, $f^{(n)}([a]) = [r]$.

The argument above shows that the structure above is a reverse search structure, so we can enumerate the classes by Algorithm 2.2 as shown in Theorem 2.3.

The adjacency function avoids self and multiple adjacency. Its time complexity becomes $\delta(\mathrm{time}(\mathrm{Adj}) + \mathrm{time}(\mathrm{rep}))$. The memory required is δ times the size of an object.

The parent function works with time complexity $\mathrm{time}(f) + \mathrm{time}(\mathrm{rep})$. □

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 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 2.6 (reverse search structure with total order)

$(S, \delta, \mathrm{Adj}, <)$ *is a* reverse search structure with total order *if*

- $(S, \delta. \mathrm{Adj})$ *satisfies conditions* (1) *to* (3) *in Definition 2.1.*
- $<$ *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 2.7

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.

Proof. We have to check the conditions (4) and (5) of Definition 2.1. By the definition of f, (4) is satisfied. The condition $f(a) = a$ holds if and only if $a = r$. For other objects, $f(a) > a$, thus there exists some $n \in \mathbb{N}$ such that $f^{(n)}(a) = r$. This shows (5). The maximum object r becomes the root. □

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

Definition 2.8 (reverse search structure for classes with total order)

$(S. \delta, \mathrm{Adj}, <, \sim)$ *is a* reverse search structure for 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 2.9

The reverse search structure for classes with total order together with

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

becomes a reverse search structure for classes.

Proof. We have to check the last condition of Definition 2.4. For any a, $f(a) \geq a$, and $f(a) = a$ if and only if $a = r$. For any a, $\mathrm{rep}(a) \geq a$. And also $\mathrm{rep}(r) = r$. Thus the root object r is the only object satisfying $(\mathrm{rep} \circ f)(r) = r$. For any other object $a \neq r$, $(\mathrm{rep} \circ f)(a) > a$, and there exists $n \in \mathbb{N}$ such that $(\mathrm{rep} \circ f)^{(n)}(a) = r$. □

3 Regular triangulations and the secondary polytope

Regular triangulations form a subset of triangulations. They correspond to the vertices of a polytope, secondary polytope, which is determined uniquely by a configuration of points. We later propose an algorithm to enumerate regular triangulation by applying a vertex enumeration method to this secondary polytope. Refer to [3, 12, 13, 19, 33] 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 for 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.

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 are intersecting properly and (2) the union of the simplices $\cup\{\sigma_1, \ldots, \sigma_m\}$ is equal to $\mathrm{conv}(\mathcal{A})$.

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 being a lower facet means, $F = \{\boldsymbol{x} \in P : \boldsymbol{c}\boldsymbol{x} = c_0\}$ is a facet with $\boldsymbol{c}\boldsymbol{x} \leq c_0$ valid for P and $c_{d+1} < 0$. 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 $\varphi_T : \mathcal{A} \to \mathbb{R}$ 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 φ_T in $\mathbb{R}^{\mathcal{A}}$ for all triangulations T of $\mathcal{A}$.

Regular triangulations 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". For the definition of flips, consult the references above.

Theorem 3.1 ([12, Chapter 7. Theorem 1.7., Theorem 2.10.])

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 appling a flip to a regular triangulation is not necessarily regular. During enumeration, we will 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.

Lemma 3.2

The number of the d-simplices and all of their faces in a regular 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 denotes 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 2.6 and Proposition 2.7. 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

Two triangulations are defined to be adjacent if they can be modified along a circuit.

Definition 4.1 (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 modified from T along a circuit)*

The index i in the definition of $\mathrm{Adj}(T, i)$ is not of importance.

Definition 4.2 (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, last condition in Definition 2.6 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 2.7.

Theorem 4.3 (enumerating regular triangulations)

The structure of Definition 4.1 and 4.2 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 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 4.6. □

4.2 Data structure for a triangulation

We represent a simplex by the set of the indices of its extreme points. For each triangulation, we hold the graph of its face poset in memory. The d-simplices used in the triangulation and their faces become the vertices of the graph. When a simplex is a face of another simplex, with their difference of dimension one, the corresponding vertices in the graph are connected by an edge. We label this edge by the index of the point included only in the larger simplex. This data structure represents the incidence relation of the simplices. It 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 an $(i+2)$-tuple $(i \leq d)$ of the indices of the points in the circuit sorted in increasing order. We maintain all of these circuits by a list of these tuples sorted in lexicographic order. For a circuit with $i+2$ points, its convex hull consists of at most $i+1$ simplices, and in practice we represent the $(i+2)$-tuple of points implicitly by recording those simplices. Any i-simplex belongs to at most $i+1$ of the circuits consisting of more than one simplices. 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. Two circuits can be compared with respect to the lexicographic ordering in $O(d)$ time by the implicit representation above. 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 the linear programming problem in $O(\mathrm{LP}(n-d-1, s))$ time, as described in Lemma 4.4 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 $O(\mathrm{LP}(n-d-1, s))$.

For points in general position, we only have to hold the graph of the d and $(d-1)$-simplices. In this case, both the space and the 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. For each $(d-1)$-simplex of a given triangulation of the set $\mathcal{A}$ of points, not on the boundary of $\mathrm{conv}(\mathcal{A})$, there are two d-simplices sharing the simplex. Suppose the two simplices have points $\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d\}$ and $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{d+1}\}$. Let w_i represent the weight of $\boldsymbol{p}_i$. If $\boldsymbol{w}$

is defining a regular triangulation, we must 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 4.4

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 seen just by setting w to the weight vector realizing the regular triangulation. The "if" part can then be shown by standard convex analysis. □

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 4.5

Whether a given triangulation is regular 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 [1, 4, 27]. [10] devises an algorithm which directly constructs a regular triangulation from an assignment of weight, while its time complexity is analyzed by means of randomized analysis since the algorithm uses flipping operation as a primitive. If one regular triangulation is necessary, this may also be used.

The lexicographically maximum one can be computed by starting with any regular triangulation and transforming it by flips towards lexicographic maximization 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 algorithm.

4.5 Complexities

Proposition 4.6

(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, with checking their regularity, and find the lexicographically maximum one. Since there are at most $O(ds)$ adjacent triangulations and each of them can be computed separately, the complexities follow. □

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 mathematical viewpoints, see [20, 21]. These are still preliminary results.

Our algorithm is implemented in C language. The experiments are done on Sun SPARCstation 10 with 64MB memory. Exact arithmetics are realized by GNU MP library for arbitrary precision integer and rational number arithmetic. Linear programming problems are solved by a 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 simplices of regular triangulations when the points are randomly generated in the d-cube with the edges of length 1000. Every coordinate is an integer less than or equal to 500 and more than -500.

- $n = d + 4$; this is, so to speak, the first non-trivial case, since in the case of $n = d + 3$ all triangulations are regular.
  - $n = 5, d = 1$: Each of 20 configurations has 8 regular triangulations.
  - $n = 6, d = 2$: 2 of 20 configurations have 16 regular triangulations, 6 of them have 15 ones, and 12 of them have 14 ones.
  - $n = 7, d = 3$: 2 of 20 configurations have 27 regular triangulations, and 18 of them have 25 ones.
  - $n = 8, d = 4$: 3 of 20 configurations have 40 regular triangulations, 3 of them have 41 ones, 7 of them have 42 ones, 3 of them 43 ones, and the other four have 44 ones.
- $n = d + 5$;
  - $n = 6, d = 1$: Each of 20 configurations has 16 regular triangulations.
  - $n = 7, d = 2$: The number of regular triangulations is quite various. 4 of 20 configurations have 42 ones, 9 of them have 46 ones, 2 of them have 50 ones, one of them has 51 ones, and 2 have 55 one, and two other configurations have 56 regular triangulations, respectively.
  - $n = 8, d = 3$: In this case the number regular triangulations varies from 128 to 168 with some small peak around 133.

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 some 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 could be obtained from partial computational results).

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. 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, let $\boldsymbol{w}_{\mathrm{D}}$ be the weight vector with $w_{D\,i} = \|\boldsymbol{a}_i\| = \sum_{j=1}^{d}(a_{i,j})^2$. 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 5.1

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, it is easy to see that

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

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

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

where $\langle w, \varphi\rangle$ is the inner product of w and φ, and $\varphi_{\Delta_{\boldsymbol{w}_{\mathrm{D}}}}$, φ_Δ 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 generalized 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 the 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 generalized 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 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 5.2

All the spanning regular triangulations can be enumerated in $O(d^2s^2\mathrm{LP}(n-d-1,s)\#\mathcal{R}_{\mathrm{spanning}})$ *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 $S = \mathcal{R}_{\mathrm{spanning}})$ 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 $\mathcal{R}_{\mathrm{spanning}})$ 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 break, we compare their volume vectors in lexicographic order. Then the claim holds similarly as Theorem 4.3: the conditions in Definition 2.6 are satisfied and the complexities are as in Proposition 4.6. □

The arguments in Lemma 5.1 can be further utilized as follows.

Theorem 5.3

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 [3], they 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 these fact, Theorem 5.3 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 5.1 and Theorem 5.3.

6 Enumeration of classes of regular triangulations

In section 2, we showed an extension of reverse search for enumeration of 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 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 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 as a set.

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 6.1 (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: $g\,\mathrm{conv}\{\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_m}\} = \mathrm{conv}\{g\boldsymbol{a}_{i_1}, \ldots, g\boldsymbol{a}_{i_m}\}$.*
- *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. We classify these sets by this equivalence classes.*

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 2.8, and the conditions for a reverse search structure for classes with order are satisfied. Theorem 2.5 leads the following.

Theorem 6.2 (enumerating classes of regular triangulations)

By reverse search structure, total order and equivalence relation defined in Definition 4.1, 4.2 and 6.1, we can enumerate the classes of regular triangulations in respect of symmetry. The time complexity and required memory are as in Theorem 2.5. Time complexities for time(Adj) *and* time(f) *are the same as the case of Theorem 4.3 and* time(rep) *in general is as in Lemma 6.3.*

Lemma 6.3

The representative of a 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

6.2.1 $\Delta_k \times \Delta_l$

We are interested in enumerating the triangulations for products of two simplices. We take as the standard d-simplex Δ_d the convex hull $\mathrm{conv}\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{d+1}\}$ 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

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

In Figure 1 we show $\Delta_1 \times \Delta_1$ and $\Delta_2 \times \Delta_1$ for example.

Our objects to enumerate are the triangulations of $\mathcal{A} = \mathrm{vert}(\Delta_k \times \Delta_l)$, where $\mathrm{vert}(\Delta_k \times \Delta_l)$ are the vertices. Examples of triangulations are shown in Figure 2.

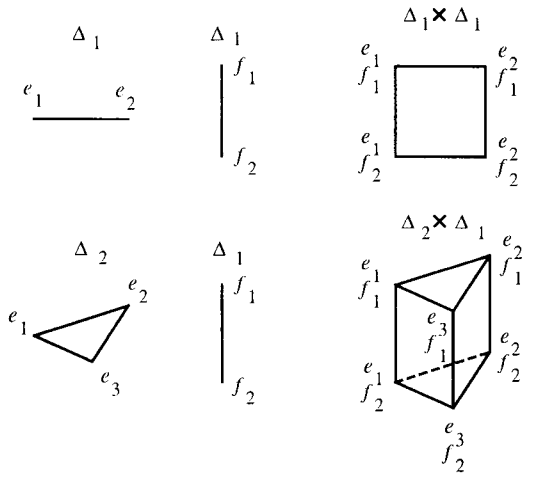

Figure 1: Product of simplices: $\Delta_1 \times \Delta_1$ and $\Delta_2 \times \Delta_1$

6.2.2 The symmetry of $\Delta_k \times \Delta_l$

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 6.4

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 $\mathrm{vert}(\Delta_k \times \Delta_l)$. Thus it suffices the conditions for the group. Actions and equivalence relations on simplices and triangulations are induced as in Definition 6.1. For example, the triangulations T_1 and T_2 in Figure2 moves to each other by $((1,2), e) \in S_2 \times S_2$. So does T_3 and T_4 by $((1,3), e) \in S_3 \times S_2$. As shown in Theorem 6.2, this equivalence relation satisfies the last condition of reverse search structure for classes with symmetry, and we can enumerate the classes of regular triangulations in respect of symmetry.

When $k = l$, there are 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.

6.2.3 Computing the representative

The volume vectors can be regarded as matrices: $(\varphi_T \begin{pmatrix} \boldsymbol{e}_i \\ \boldsymbol{f}_j \end{pmatrix})_{ij} \in \mathbb{R}^{k+1} \times \mathbb{R}^{l+1}$. Those corresponding to the triangulations in Figure2 are

$$\varphi_{T_1} = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \quad \varphi_{T_2} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \quad \varphi_{T_3} = \begin{pmatrix} 2 & 2 \\ 3 & 1 \\ 1 & 3 \end{pmatrix} \quad \varphi_{T_4} = \begin{pmatrix} 1 & 3 \\ 3 & 1 \\ 2 & 2 \end{pmatrix}.$$

$S_{k+1} \times S_{l+1}$ acts on a volume vector φ_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

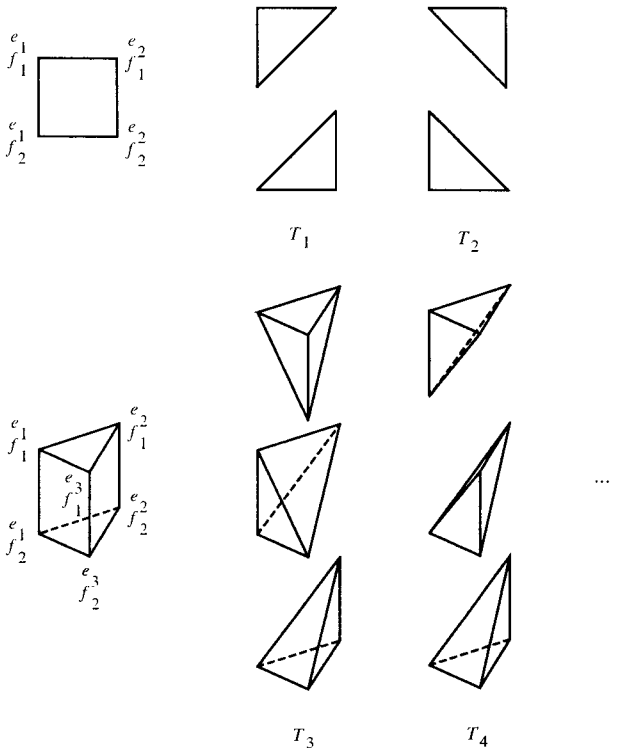

Figure 2: Triangulations for $\Delta_1 \times \Delta_1$ and $\Delta_2 \times \Delta_1$

φ_T and $\varphi_{T'}$ are in the same class, since the correspondence between regular triangulations and volume vectors was one-to-one (cf. Theorem 3.1).

We introduced a total order on the regular triangulation by lexicographic order of their corresponding volume vectors (Definition 4.2). For the case of $\Delta_k \times \Delta_l$, we can regard this order as 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. In Figure 2, T_2 becomes the representative of the class $\{T_1, T_2\}$.

Lemma 6.5

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

Proof. In order to choose the representative triangulation from the class of a given regular triangulation, we look for an element of $S_{k+1} \times S_{l+1}$ whose corresponding rearrangement maximizes the matrix of the volume vector φ_T. We check all of the $(l+1)!$ arrangements of columns. For each of them, the maximum can be obtained by sorting the rows. There are $k+1$ rows of length $l+1$. Comparing two numbers in unit time, a comparison between two rows takes $l+1$ time. So we can sort the rows in $O((k+1)^2(l+1))$ time. Hence the whole time complexity is $O((l+1)!\,(k+1)^2(l+1)) = O(l!\, k^2 l^2)$. □

This is faster than the time complexity for the general case in Lemma 6.3.

6.2.4 Enumerating classes of regular triangulations

Proposition 6.6

The enumeration algorithm for classes of regular triangulations in Theorem 6.2, 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.

Proof. The time to compute the representative element time(rep) is $O(l!\, k^2 l^2)$ by Lemma 6.5. The $l!$ here appears in the whole time complexity, but since we are just finding the maximum arrangement of a small matrix (remind the instances we have to solve are for $k, l = 3$ or 4), practically this is not time consuming compared to solving LPs for each elements in a class. □

6.3 Hypercubes

6.3.1 C_d

We are interested in enumerating the triangulations for hypercubes. We write $\boldsymbol{e}_i$ for unit vectors with the i-th element one and the rest zeros. The d-cube C_d is given by

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

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

6.3.2 The symmetry of C_d

We define the symmetry of C_d as follows.

Definition 6.7

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 the 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 $\mathrm{vert}(C_d)$. Thus it suffices the conditions for the group. Actions and equivalence relations on simplices and triangulations are induced as in Definition 6.1. As shown in Theorem 6.2, this equivalence relation satisfies the last condition of reverse search structure for classes with symmetry, and we can enumerate the classes of regular triangulations in respect of symmetry.

6.3.3 Enumerating classes of regular triangulations

Proposition 6.8

The enumeration algorithm for classes of regular triangulations in Theorem 6.2, 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

In section 8, we will 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 [18, 32]. We reformulate one of these algorithms for our case, and later propose a triangulation enumerating algorithm.

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 [18]. 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 will correspond to all of the d-simplices, which can become large.

We reformulate algorithm [18] 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 for some degree compared to just applying [18] to graphs.

7.1 Enumerating maximal independent sets

Here, we show the maximal independent set enumeration algorithm from [18], on which our algorithm is based. It is called the generalized Paull-Unger procedure [24] with improvements by Tsukiyama, Ide, Ariyoshi and Shirakawa [32].

Let the base set be $E = \{1, \ldots, n\}$, c the independence testing time, 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 reversely: first the maximal independent sets in $I \cup \{j\}$ are listed up, and then their maximal independence in $\mathcal{M}_j$ is checked. The algorithm elaborates to produce I' from a single I.

Algorithm 7.1 (enumerating maximal independent sets [18])

Step 1. *For each $I \in \mathcal{M}_{j-1}$, find all independent sets I' that are maximal within $I \cup \{j\}$.*
Step 2. *For each such I', test I' for maximality within $\{1, \ldots, j\}$. Each set I' that is maximal within $\{1, \ldots, j\}$ is a member of $\mathcal{M}_j$, and each member of $\mathcal{M}_j$ can be found in this way. However a given $I' \in \mathcal{M}_j$ may be obtained from more than one $I \in \mathcal{M}_{j-1}$. In order to eliminate duplications we need one further step.*
Step 3. *For each I' obtained from $I \in \mathcal{M}_{j-1}$ that is maximal within $\{1, \ldots, j\}$, test for each $i < j$, $i \notin I$. the set $(I' \setminus \{j\}) \cup (I \cap \{1, \ldots, i-1\}) \cup \{i\}$ for independence. Reject I' if any of these tests yields an affirmative answer. (This step retains I' only if it is obtained from the lexicographically smallest $I \in \mathcal{M}_{j-1}$.)*

This 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, and the path from the root to the current I needs to be memorized.

Theorem 7.2 ([18])

Algorithm 7.1 enumerates all maximal independent sets in $O((nc' + n^2cK')\#\mathcal{M})$ time and $O(nK')$ memory. We suppose that in **Step 1**, *for each $I \in \mathcal{M}_{j-1}$, at most K' sets I' are found in c' time.*

7.2 Enumerating maximal independent sets of a graph

Next, we reformulate Algorithm 7.1 above 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 for a given vertex for some fixed order of vertices. This seems trivial when we write $E = \{1, \ldots, n\}$, but it is not for our case, because the vertices will correspond to the d-simplices $\mathcal{S}$. The existence of such 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.

The time complexity of Theorem 7.2 is given by c and c'. The time for an independence test was c. For any set $I \subset E$, this test can be done by checking if any pair of vertices in I are connected by an edge. If such pair exists, I is dependent, and if not, independent. This takes $(\#I)^2$ time(intersect) time, m^2 time(intersect) at most, which corresponds to the c in Theorem 7.2. However, by the following realization it can be done in m time(intersect) time.

Algorithm 7.3 (enumerating maximal independent sets of a graph)

We reformulate Algorithm 7.1 as follows.

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 with any of the vertices in I, $I \cup \{j\}$ is the only maximal independent set in $I \cup \{j\}$. Further, this is maximal independent in $\{1, \ldots, j\}$. So, the test in* **Step 2** *is not necessary for this case.*

(2) *If j intersects with some of the vertices in I, I and the set $\{i \in I \cup \{j\} :$ not intersecting with $j\}$ are the maximal independent sets of $I \cup \{j\}$. Further, I is maximal independent in $\{1, \ldots, j\}$, so the test in* **Step 2** *is unnecessary for this. For $\{i \in I \cup \{j\} :$ not intersecting with $j\}$, we need the test.*

Step 2. *We check the maximality of I' in $\{1, \ldots, j\}$. As mentioned above, the only case we have to check is the second candidate in case* (2) *of* **Step 1**. *We check whether some $i \in \{1, \ldots, j\} \setminus I'$ is not intersecting with all the vertices in I'. If such i exists, I' is not maximal independent, and if not, it is maximal independent.*

Step 3. *We check whether I' is obtained from the lexicographically smallest $I \in \mathcal{M}_{j-1}$. This is always true for case* (1) *and for the first candidate in case* (2) *of* **Step 1**. *So, we only have to check for the second candidate in case* (2).

For each I' 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 this independence test can be done by checking whether i intersects with some vertex in $(I' \setminus \{j\}) \cup (I \cap \{1, \ldots, i-1\})$. If i intersects with some vertex, $(I' \setminus \{j\}) \cup (I \cap \{1, \ldots, i-1\}) \cup \{i\}$ is not independent. If i does not intersect with 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**, c' in Theorem 7.2 is computed in m time(intersect) time. For each I in $\mathcal{M}_{j-1}$ the candidates I' we take are one or two, so $K' \leq 2$. The total time complexity for this step is $O(nc'\#\mathcal{M}) = O(m\,\text{time(intersect)}n\#\mathcal{M})$.

In **Step 2**, each independence test can be done in $c = m$ time(intersect) time, and it takes m time(intersect)n time for each I'. The total time complexity for this step is $O(m\,\text{time(intersect)}\ n^2\#\mathcal{M})$.

In **Step 3**, the independence test can be done in $c = m$ time(intersect) time. The total time complexity for this step is $O(m\,\text{time(intersect)}n^2\#\mathcal{M})$.

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 first candidate for case (2) in **Step 1**, 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. Remind 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 second candidate for case (2) in **Step 1**.

The time complexity needed for a recomputation of I is m time(intersect)n and $O(m$ 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 memorize the path from the root to the current node, which became the space complexity $O(nK')$ in Theorem 7.2. This saving of memory was implied in [18]. We realize this 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 will try the second candidate $\{i \in I \cup \{j\} : \text{ not intersecting with } j\}$, if it is a child, when we come back to this node ascending from $I \in \mathcal{M}_j$.

When we are ascending the tree, there are three possibilities. We came from the only child for case (1), the first or the second for case (2). This was the information of the path from the root, which took $O(nK')$. We can dispense with this as shown above in the analysis ascending the tree.

Since we have the oracle mentioned above, we do not have to memorize 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 7.4

Algorithm 7.3 enumerates all maximal independent sets of a simple undirected graph. This works in $O(m\,\text{time(intersect)}n^2\#\mathcal{M})$ time 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 the vertices and edges between those 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

We were given a configuration of points $\mathcal{A} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\} \subset \mathbb{R}^d$, with their convex hull $\mathrm{conv}(\mathcal{A})$ having full dimension d. We are interested in the 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 for 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 are intersecting 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 $\mathcal{S}$.

We define the intersection graph as follows.

Definition 8.1 (intersection graph)

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

Several classes of sets of d-simplices are defined.

Definition 8.2

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

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. Further, 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 8.3

$\mathcal{T}$ 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 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. [23, 10.2.1], [26]). 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.

8.2 Enumerating triangulations

Now we apply our formulation above towards the enumeration of triangulations. The base set E is the set of d-simplices $\mathcal{S}$. We supposed the existence of an oracle which answers 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 8.4 (enumerating triangulations)

Using Algorithm 7.3, we can enumerate all maximal independent sets, thus the triangulations, of the intersection graph of $\mathcal{S}$. This works in $O(m\,\mathrm{time(intersect)}(\#\mathcal{S})^2\#\mathcal{M})$ 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, we can say that the required memory is only twice the size of a triangulation.

8.3 Enumerating d-simplices

We supposed the existence of an oracle which answers 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 was $\mathcal{A} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$. A set of $d+1$ points $\{\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_{d+1}}\}$ becomes the set of vertices for a d-simplex if and only if $\begin{pmatrix}\boldsymbol{a}_{i_1}\\1\end{pmatrix}, \ldots, \begin{pmatrix}\boldsymbol{a}_{i_{d+1}}\\1\end{pmatrix}$ are linearly independent.

Thus the problem reduces to the existence of a similar oracle for the bases of $\left\{\begin{pmatrix}\boldsymbol{a}_1\\1\end{pmatrix}, \ldots, \begin{pmatrix}\boldsymbol{a}_n\\1\end{pmatrix}\right\}$. This can be realized by reverse search with the time complexity $O((d+1)n\#\mathcal{S})$ for the whole enumeration [2]. For a given base, answering its next or previous base can be done in approximately $O((d+1)n)$ time.

Using this oracle, we do not need to memorize all of the d-simplices, thus memory for only several times the size of a simplex would be enough. This enables handling of large size problems. For such large size problems that even the memory for simplices matters, the enumeration of all triangulations may be hopeless, since it may require enormous time. However, since the algorithm does work, we can generate several triangulations among the whole.

For smaller problems for which we can memorize all of the d-simplices, it is better to enumerate and memorize them. This will be much faster than asking the oracle each time. The enumeration here can be done by reverse search as mentioned above. Though practically, 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 8.4 was evaluated by the number of this calculation. Here, we give algorithms and their complexity for this calculation. (A matrix will be regarded as a set of column vectors.)

Algorithm 8.5 (testing the intersection of d-simplices)

Input: $\{\boldsymbol{p}_1,\ldots,\boldsymbol{p}_{d+1}\}$, $\{\boldsymbol{q}_1,\ldots,\boldsymbol{q}_{d+1}\}$: *vertices of d-simplices in* $\mathbb{R}^d$
Output: *whether the simplices are intersecting improperly or not*
Suppose $\{\boldsymbol{p}_1,\ldots,\boldsymbol{p}_{d+1}\}\cap\{\boldsymbol{q}_1,\ldots,\boldsymbol{q}_{d+1}\}=\emptyset$. *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* $\mathrm{conv}\{\boldsymbol{0}\ ,\boldsymbol{e}_1,\ldots,\boldsymbol{e}_d\}$ *if and only if these simplices are intersecting improperly. This is equivalent to whether the linear programming*

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

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

under some cost vector has a feasible solution or not. When $\{\boldsymbol{p}_1,\ldots,\boldsymbol{p}_{d+1}\}$ *and* $\{\boldsymbol{q}_1,\ldots,\boldsymbol{q}_{d+1}\}$ *have points in common, the testing reduces to smaller linear programmings, after neglecting by projection the dimensions spanned by the points in common.*

Lemma 8.6

Algorithm 8.5 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 memorize whether the simplices are intersecting properly or improperly for all pairs of simplices, it is better to compute first all the intersections and memorize them. This requires memory of $\binom{\#\mathcal{S}}{2}$ bits. It can be done in time(intersect)$\binom{\#\mathcal{S}}{2}$ time. 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 preprocess we can remove away the factor time(intersect) of $\mathcal{M}$ of the time complexity in Theorem 8.4 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 the following.

Lemma 8.7

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 $K_{k+1,l+1}$ [12, 7.3.D.]. This derives the next.

Lemma 8.8

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 [17, 28]. Thus we can generate the corresponding $(k+l)$-simplices similarly.

Lemma 8.9 (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 supposed for Algorithm 7.3 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 $K_{k+1,l+1}$ [6, Lemma 2.3.], which leads the time complexity. The intersection test for this $\Delta_k \times \Delta_l$ case can be computed faster using this graph property compared to Algorithm 8.5.

Lemma 8.10 (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* $O(k+l)$ *time.*

We apply Theorem 8.4 to the case of $\Delta_k \times \Delta_l$.

Theorem 8.11 (enumerating triangulations for $\Delta_k \times \Delta_l$)

For the point configuration $\mathcal{A} = \mathrm{vert}(\Delta_k \times \Delta_l)$*, Algorithm 7.3 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 8.9, the oracle exists. By Lemma 8.7, and $m = \binom{k+l}{k}$. By Lemma 8.8, $\#\mathcal{S} = (k+1)^l(l+1)^k$. By Lemma 8.10, time(intersect) $= O(k+l)$. □

Acknowledgments

We would like to thank Nobuki Takayama of Kobe University for introducing us these problems and giving us lectures on many mathematical aspects of them. The work of the third author was supported in part by the Grant-in-Aid of the Ministry of Education, Japan.

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

References

[1] David Avis and Komei Fukuda, *A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra*, Discrete Comput. Geom., 8 (1992), 295–313.

[2] David Avis and Komei Fukuda, *Reverse search for enumeration*, Discrete Appl. Math., 65 (1996) 21–46.

[3] Louis J. Billera, Paul Filliman, and Bernd Sturmfels, *Constructions and complexity of secondary polytopes*, Advances in Math., 83 (1990), 155–179.

[4] B. Chazelle, *An optimal convex hull algorithm and new results on cuttings*, in Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science, 1991, 29–38.

[5] Jesús A. de Loera, *Computing triangulations of point configurations*, Preprint, December 1994.

[6] Jesús A. de Loera, *Nonregular triangulations of products of simplices*, Discrete Comput. Geom., 15 (1996), 253–264.

[7] Jesús A. de Loera, Serkan Hoşten, Francisco Santos, and Bernd Sturmfels, *The polytope of all triangulations of a point configuration*, Doc. Math., 1 (1996), 103–119.

[8] Jesús A. de Loera, Bernd Sturmfels, and R. R. Thomas, *Gröbner bases and triangulations of the second hypersimplex*, Combinatorica, 15 (1995), 409–424.

[9] Herbert Edelsbrunner, *Algorithms in combinatorial geometry*, Springer-Verlag, 1987.

[10] Herbert 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] Israel M. Gelfand, Mikhail M. Kapranov, and Andrei V. Zelevinsky, *Discriminants, resultants and multidimensional determinants*, Birkhäuser, Boston 1994.

[13] Israel M. Gel'fand, Andrei V. Zelevinskiĭ, and Mikhail M. Kapranov, *Newton polyhedra of principal A-determinants*, Soviet Math. Dokl., 40 (1990), 278–281.

[14] Hiroshi Imai and Keiko Imai, *Triangulation and convex polytopes*, RIMS Kokyuroku 934 (1996), Research Institute for Mathematical Sciences, Kyoto University, 149–166 (in Japanese).

[15] Barry Joe, *Three-dimensional triangulations from local transformations*, SIAM J. Sci. Stat. Comput., 10 (1989), 718–741.

[16] Barry Joe, *Construction of three-dimensional Delaunay triangulations using local transformations*, Computer Aided Geometric Design, 8 (1991), 123–142.

[17] Sanjiv Kapoor and H. Ramesh, *Algorithms for enumerating all spanning trees of undirected and weighted graphs*, SIAM J. Comput., 24 (1995) 247–265.

[18] 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.

[19] Carl W. Lee, *Regular triangulations of convex polytopes*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 4, Amer. Math. Soc. 1991, 443–456.

[20] TOMONARI 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.

[21] TOMONARI MASADA, *An algorithm for the enumeration of regular triangulations*, Master's Thesis, Department of Information Science, University of Tokyo, March 1995.

[22] TOMONARI MASADA, HIROSHI IMAI, AND KEIKO IMAI, *Enumeration of regular triangulations*, in Proc. 12th Annual ACM Symposium on Computational Geometry, New York 1996, 224–233.

[23] JOSEPH O'ROURKE, *Art gallery theorems and algorithms*, Oxford University Press, New York, 1987.

[24] M. C. PAULL AND S. H. UNGER, *Minimizing the number of states in incompletely specified sequential switching functions*, IRE Trans. Electronic Computers, 1959, 356–367.

[25] GÜNTER ROTE, *Degenerate convex hulls in high dimensions without extra storage*, in Proc. 8th Annual ACM Symposium on Computational Geometry, 1992, 26–32.

[26] E. SCHÖNHARDT, *Über die Zerlegung von Dreieckspolyedern in Tetraeder*, Math. Ann., 98 (1928), 309–312.

[27] RAIMUND SEIDEL, *Constructing higher-dimensional convex hulls at logarithmic cost per face*, in Proc. 18th Annual ACM Symposium on the Theory of Computing, 1986, 404–413.

[28] AKIYOSHI SHIOURA, AKIHISA TAMURA, AND TAKEAKI UNO, *An optimal algorithm for scanning all spanning trees of undirected graphs*, SIAM J. Comp., 26 (1997), 678–692.

[29] BERND STURMFELS, *Gröbner bases of toric varieties*, Tôhoku Math. J. 43 (1991), 249–261.

[30] BERND STURMFELS, *Gröbner bases and convex polytopes*, American Mathematical Society, 1996.

[31] FUMIHIKO TAKEUCHI AND HIROSHI IMAI, *Enumerating triangulations for products of two simplices and for arbitrary configurations of points*, in Proc. Computing and combinatorics: 3rd annual international conference, Lecture Notes in Computer Science 1276, Springer-Verlag, Berlin, 470–481.

[32] SHUJI TSUKIYAMA, MIKIO IDE, HIROMU ARIYOSHI, AND ISAO SHIRAKAWA, *A new algorithm for generating all the maximal independent sets*, SIAM J. Comput., 6 (1977), 505–517.

[33] GÜNTER M. ZIEGLER, *Lectures on polytopes*, Springer-Verlag, New York, 1995.

[34] `http://naomi.is.s.u-tokyo.ac.jp`