Enumerating Triangulations in General Dimension

by

Fumihiko Takeuchi

A Master's Thesis

Submitted to

The Graduate School of Information Science
Faculty of Science
The University of Tokyo
on February 5, 1998
(revised, English version: March 19, 1998)
in Partial Fulfillment of the Requirements
for the Degree of Master of Science

Thesis Supervisor: Hiroshi Imai
Title: Associate Professor of Information Science

ABSTRACT

The aim of this research is to study triangulations in general dimension through enumeration. Many results have been achieved for triangulations in dimension two. For triangulations in general dimension, regular triangulations, which form a subclass of all triangulation, have been studied recently. However nonregular triangulations are not yet well understood

We first propose an algorithm to enumerate all triangulations for arbitrary configurations of points. Regular triangulations form a nice algebraic structure, and algorithms to enumerate them efficiently is known. However, no such thing was known for all triangulations. We accomplish this by characterizing triangulations as maximal independents sets of an intersection graph, and enumerating the maximal independent sets. The intersection graph here, is a graph with vertices the maximal dimensional simplices of the given point configuration, and edges between improperly intersecting simplices. Thus triangulations form a subset of the maximal independent sets of this graph.

We next propose an algorithm to enumerate efficiently the regular triangulations of highly symmetric polytopes. Among those we are interested in triangulations of products of two simplices and hypercubes. It is not efficient to enumerate naively the triangulations of these symmetric polytopes, because we may count the "same" triangulation many times. We accomplish this by enumerating classes of triangulations in respect of symmetry. This is done by introducing reverse search for classes of objects.

We also consider facets of independent set polytopes of intersection graphs of simplices. We deal with two intersection graphs. The intersection graph of d-simplices and the graph of $(d-1)$-simplices for point configurations having dimension d. The independent set polytopes are the convex hulls of the incidence vectors of the independent sets of these graphs. As a special case, we deal with points spanning the plane. We give a proof that an inequality known to be powerful for the minimum weight triangulation problem is defining a facet of the independent set polytope.

Acknowledgements

I would like to thank my thesis supervisor Dr. Hiroshi Imai, secretary of our laboratory Minako Okada, librarian of our department Akiko Shintani, my colleagues of the laboratory, my friends and my parents for their helpful advice, assistance, discussion and encouragement.

Table of Contents

Chapter 1

Enumeration of triangulations

We propose an algorithm to enumerate all triangulations, regular or not, for arbitrary configurations of points in general dimension. There are many results for triangulations in two dimension, but little is known for higher dimensions. Algorithms to enumerate regular triangulations are studied well, however, no efficient algorithm to enumerate all triangulations, including nonregular ones, has been known. Our algorithm handles this problem for arbitrary configurations of points. It formulates triangulations as maximal independent sets of an intersection graph, and is based on a general maximal independent set enumeration algorithm. The intersection 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.

1.1 Introduction

Gel'fand, Kapranov and Zelevinsky introduced the secondary polytope for point configurations in general dimensional space, and showed that its vertices correspond to regular triangulations [7], [8]. Using this property, regular triangulations can be enumerated by enumerating the vertices of that polytope. Billera, Filliman and Sturmfels studied the structure of this secondary polytope with relation to volume vectors, and analyzed the complexity computing the polytope. They also proposed the universal polytope, in which the vertices correspond to all triangulations, but it has not resulted in a practical triangulation enumeration method [3].

We have been interested in the triangulation of products of two simplices $\Delta_k \times \Delta_l$, where k and l are their dimensions. Their regular triangulations have relations with other branches of mathematics, such as Gröbner bases [21], [22]. De Loera devised a program to enumerate regular triangulations for given sets of points. He enumerated the triangulations, all of which are regular, for $\Delta_2 \times \Delta_3$ and $\Delta_2 \times \Delta_4$ [4], [5].

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 [6], 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, and apply a general maximal independent set enumeration algorithm. 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. [18, 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.

Finally, 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.

We first define the intersection graph (Section 2), and enumerate triangulations as maximal independent sets of this graph (Section 3). We discuss further two basic operations used in the enumeration: the enumeration of maximal dimensional simplices (Section 4) and testing whether two simplices are intersecting improperly or not (Section 5). We also show the enumeration for products of two simplices in which case parts of the algorithm can be made faster (Section 6).

1.2 Triangulations as maximal independent sets

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

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

We define the intersection graph as follows.

Definition 1.1 (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 1.2

- $\mathcal{I} = \{I \in 2^{\mathcal{S}} :$ 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 $S\}$

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 intersection, I is an maximal independent set. This gives the following proposition.

Proposition 1.3 ([9])

$\mathcal{T}$ 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 $\mathrm{conv}(S)$.

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. [18, 10.2.1]). 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.

1.3 Enumeration of triangulations

As in the previous section, triangulations can be regarded as a subclass of the maximal independent sets of the intersection graph of d-simplices. Efficient algorithms to enumerate maximal independent sets are known [13], [24]. We reformulate one of these algorithms to our case, and propose a triangulation enumerating algorithm. This algorithm handles arbitrary configurations of points.

The algorithm we use to enumerate maximal independent sets is from [13]. It is called the generalized Paull-Unger procedure with improvements by Tsukiyama, Ide, Ariyoshi and Shirakawa [24].

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 independency 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 1.4 (enumeration of maximal independent sets [13])

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 performs 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 1.5 ([13])

Algorithm 1.4 enumerates all maximal independent sets in $O(nc'K + n^2cKK')$ time and $O(nK')$ memory. Here $K = \#\mathcal{M}$ and we suppose that in **Step 1**, for each $I \in \mathcal{M}_{j-1}$, at most K' sets I' are found in c' time.

Next, we reformulate the algorithm above for enumeration of maximal independent sets of an simple undirected graph. Our aim was to enumerate the maximal independent sets of the intersection graph of d-simplices.

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 correspond to the d-simplices $\mathcal{S}$. The existence of such oracle is discussed in Section 1.4.

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 1.5 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, $m^2 \cdot$ time(intersect) at most, which corresponds to c in Theorem 1.5. However, by the following realization it can be done in $m \cdot$ time(intersect) time.

Algorithm 1.6 (enumeration of maximal independent sets of graphs)

We reformulate Algorithm 1.4 as follows.

Step 1. For each I in $\mathcal{M}_{j-1}$ we want to find the candidates I'. 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 have to 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. It is checked whether I' is obtained from the lexicographically smallest $I \in \mathcal{M}_{j-1}$. This is always true for case (1) and 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 1.5 is computed in $m \cdot$ 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 $O(nc'K) = O(m\, \text{time(intersect)} n \# \mathcal{M})$.

In **Step 2**, each independence test can be done in $c = m \cdot \text{time(intersect)}$ time, and it takes $m\,\text{time(intersect)}n$ 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 $c = m \cdot \text{time(intersect)}$ 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 only $2m$ memory is allowed, we have to recompute the parent when ascending the tree. However, this does not increase the order of 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 (last element of $I' \setminus \{j+1\}) + 1, \ldots, j$ 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 $m\,\text{time(intersect)}n$ and $O(m\,\text{time(intersect)}n^2\#\mathcal{M})$ as a whole. Thus ascending the tree does not increase the order of the time 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 1.5. This saving of memory was implied in [13]. 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 1.7

Algorithm 1.6 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 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 $\{1,\dots,j\}$, and performing searches for each subtrees with roots $I \in \mathcal{M}_j$.

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 Section 1.4. 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 1.8 (enumeration of triangulations)

Using Algorithm 1.6, we can enumerate all maximal independent sets, thus the triangulations, of the intersection graph of $\mathcal{S}$. This works in $O(m\,\text{time(intersect)}(\#\mathcal{S})^2\#\mathcal{M})$ 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.

1.4 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 $S = \{\boldsymbol{p}_1,\dots,\boldsymbol{p}_n\}$. A set of $d+1$ points $\{\boldsymbol{p}_{i_1},\dots,\boldsymbol{p}_{i_{d+1}}\}$ becomes the set of vertices for a d-simplex if and only if $\binom{\boldsymbol{p}_{i_1}}{1},\dots,\binom{\boldsymbol{p}_{i_{d+1}}}{1}$ are linearly independent.

Thus the problem reduces to the existence of a similar oracle for the bases of $\{\binom{\boldsymbol{p}_1}{1},\dots,\binom{\boldsymbol{p}_n}{1}\}$. This can be realized by reverse search with the time complexity $O((d+1)n\#\mathcal{S})$ for the whole enumeration [2].

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 also by reverse search. 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.

1.5 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 1.8 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 1.9 (testing the intersection of d-simplices)

Input: $\{\boldsymbol{a}_1, \dots, \boldsymbol{a}_{d+1}\}$, $\{\boldsymbol{b}_1, \dots, \boldsymbol{b}_{d+1}\}$: vertices of d-simplices in $\mathbb{R}^d$
Output: whether the simplices are intersecting improperly or not
Suppose $\{\boldsymbol{a}_1, \dots, \boldsymbol{a}_{d+1}\} \cap \{\boldsymbol{b}_1, \dots, \boldsymbol{b}_{d+1}\} = \emptyset$. First, by affine transformation, we move $(\boldsymbol{b}_1 \cdots \boldsymbol{b}_{d+1})$ to $(\boldsymbol{0}\ \boldsymbol{e}_1 \cdots \boldsymbol{e}_d)$, where $\boldsymbol{e}_i$ are the unit vectors. The points $(\boldsymbol{a}_1 \cdots \boldsymbol{a}_{d+1})$ move to $(\boldsymbol{b}_2 - \boldsymbol{b}_1 \cdots \boldsymbol{b}_{d+1} - \boldsymbol{b}_1)^{-1}(\boldsymbol{a}_1 - \boldsymbol{b}_1 \cdots \boldsymbol{a}_{d+1} - \boldsymbol{b}_1)$. 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 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 $\{\boldsymbol{a}_1, \dots, \boldsymbol{a}_{d+1}\}$ and $\{\boldsymbol{b}_1, \dots, \boldsymbol{b}_{d+1}\}$ have points in common, the testing reduces to smaller linear programmings, after neglecting by projection the dimensions spanned by the points in common.

Lemma 1.10

Algorithm 1.9 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) $\cdot \binom{\#\mathcal{S}}{2}$ 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 1.8 to $O(\text{time(intersect)}(\#\mathcal{S})^2 + m(\#\mathcal{S})^2\#\mathcal{M})$.

1.6 $\Delta_k \times \Delta_l$

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

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

In Figure 1.1 we show $\Delta_1 \times \Delta_1$ and $\Delta_2 \times \Delta_1$ for example.

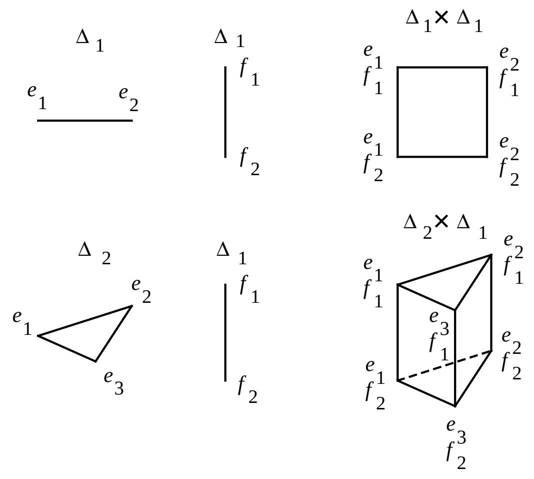

Figure 1.1: Product of simplices: $\Delta_1 \times \Delta_1$ and $\Delta_2 \times \Delta_1$

We want to enumerate the triangulations for the point configuration $S = \operatorname{vert}(\Delta_k \times \Delta_l)$, where $\operatorname{vert}(\Delta_k \times \Delta_l)$ are the vertices. Examples of triangulations are shown in Figure 1.2.

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 1.11

The number of $(k+l)$-simplices included in a triangulation of $\Delta_k \times \Delta_l$ is $(k+l)!/k!\,l!$.

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, 7.3.D.]. This derives the next.

Lemma 1.12

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 [10] [20]. Thus we can generate the corresponding $(k+l)$-simplices similarly.

Lemma 1.13 (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 1.6 exists.

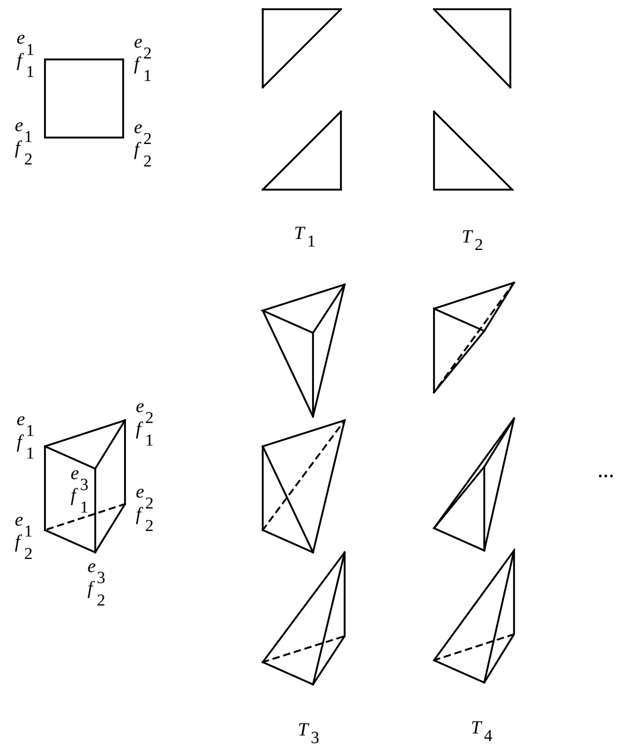

Figure 1.2: Triangulations for $\Delta_1 \times \Delta_1$ and $\Delta_2 \times \Delta_1$

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 $K_{k+1,l+1}$ [5, 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 1.9.

Lemma 1.14 (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.

1.6.1 Enumerating triangulations for $\Delta_k \times \Delta_l$

We apply Theorem 1.8 to the case of $\Delta_k \times \Delta_l$.

Theorem 1.15

For the point configuration $S = \text{vert}(\Delta_k \times \Delta_l)$, Algorithm 1.6 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 $2m$.

Proof. By Lemma 1.13, the oracle exists. By Lemma 1.11, and the consideration of the volume of $(k+l)$-simplices above it, maximal independent sets have cardinality at most $m = \binom{k+l}{k}$. By Lemma 1.12, $\#\mathcal{S} = (k+1)^l(l+1)^k$. By Lemma 1.14, time(intersect) $= O(k+l)$. $\square$

1.7 Future works

1.7.1 Enumeration of classes of triangulations

Applying this method to enumerate the classes of triangulations with respect to symmetry would be a future work. Suppose we defined two triangulations or maximal independent sets to be equivalent when they can be regarded the same with respect to the symmetry of the given point configuration. We want to enumerate the classes of maximal independent sets with respect to this symmetry. The problem becomes to enumerate the equivalence classes of maximal independent sets of graphs. However, this seems to be difficult.

Chapter 2

Enumeration of classes of regular triangulations

We propose an algorithm to enumerate classes of objects by reverse search. We apply this to the enumeration of classes of regular triangulations in respect of symmetry for symmetric polytopes. Application to products of two simplices and hypercubes are shown. There are many results for triangulations in two dimension, but little is known for higher dimensions. The objects we enumerate in this paper are for general dimensions.

Products of two simplices and hypercubes are polytopes rather simple, but their triangulations are not yet well understood. Since these polytopes are highly symmetric, counting all triangulations naively is inefficient: we may count the "same" triangulation many times. Our algorithm enumerates the classes of regular triangulations, a subset of triangulations, with respect to the symmetry. We use reverse search technique, utilizing the symmetric structure of the polytope. This enables time complexity linear to the number of these classes, and space complexity of the size of several triangulations.

2.1 Introduction

Gel'fand, Kapranov and Zelevinsky introduced the secondary polytope for point configurations in general dimensional space, and showed that its vertices correspond to regular triangulations [7], [8]. Using this property, regular triangulations can be enumerated by enumerating the vertices of that polytope. Billera, Filliman and Sturmfels studied the structure of this secondary polytope with relation to volume vectors, and analyzed the complexity computing the polytope [3].

Regular triangulations of the product of two simplices $\Delta_k \times \Delta_l$, where k and l are their dimensions, have relations with other branches of mathematics, such as Gröbner bases [21], [22]. This polytope is highly symmetric: it has the symmetry of the direct product of two symmetric groups $S_{k+1} \times S_{l+1}$. So, it is not smart to count all triangulations naively, because we may count the "same" one $(k+1)!\,(l+1)!$

times. De Loera devised a program to enumerate regular triangulations for given sets of points. The 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 $\Delta_2 \times \Delta_4$ [4], [5]. 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.

Masada, Imai and Imai proposed an algorithm to enumerate regular triangulations with output-size sensitive time complexity, which is same as de Loera's, using the memory only of the size for two triangulations [16], [17]. It uses a general technique for enumeration which is called reverse search, by Avis and Fukuda [1], [2].

We first 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. As mentioned above, for these highly symmetric polytopes, it is important to enumerate the classes of triangulations. Using reverse search, we imaginary make a tree with the classes the vertices, and enumerate those vertices traversing the tree only by using local information. The algorithm runs in output-sensitive time, i.e. in time proportional to the number of classes, and requires memory only several times of a triangulation.

We begin by a brief explanation of reverse search (Section 2). Next we give our formulation of reverse search for classes of objects (Section 3). We introduce results on regular triangulations (Section 4) and their enumeration (Section 5). Then we present our algorithm for enumeration of classes of regular triangulations (Section 6). Applications to products of simplices (Section 7) and hypercubes (Section 8) are shown.

2.2 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 for twice the size of an object among those we want to enumerate. 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 enumerate. 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 we do not need r, because we can find it by applying f several times to an object.

Algorithm 2.2 (reverse search [2])

```
**ReverseSearch**($\delta$, 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 time(Adj) and time(f) are the time necessary to compute functions Adj and f. The memory required is twice the size of an object in S.

2.3 Reverse search for classes

Later, we will show an algorithm to enumerate classes of regular triangulations. This will be based on the enumeration of classes of objects by reverse search we propose in this section.

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

The time complexity is $O(\delta(\delta(\mathrm{time(Adj)}+\mathrm{time(rep)})+\mathrm{time}(f))\#(S/\!\sim))$ 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 for 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

$d \sim \text{rep}(a)$ adjacent to $\text{rep}(b)$. Thus $\text{Adj}(\text{rep}(b), j) = d$ for some j, and $[\text{rep}(b)] = [c] \neq [\text{rep}(a)] = [d]$. This implies $\text{Adj}([\text{rep}(b)], k) = [d] = [\text{rep}(a)]$ for some k.
(1), (2) and (3) are satisfied by definition.
(4): if $f([a]) = [b]$ and $[a] \neq [b]$, $f(\text{rep}(a)) \sim b \not\sim a$. Thus $f(\text{rep}(a)) \not\sim \text{rep}(a)$. Since $f(\text{rep}(a)) \neq \text{rep}(a)$, $\text{Adj}(\text{rep}(a), i) = f(\text{rep}(a))$ for some i. Thus $\text{Adj}([\text{rep}(a)], j) = [f(\text{rep}(a))]$ for some j. This leads $\text{Adj}([a], j) = [b]$.
First statement of condition (5): the only class $[a]$ with $f([a]) = [a]$ is the class which includes the original root object r. Since $r = \text{rep}(f(r))$, 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 $[f(\text{rep}(a))] = [a]$, thus $\text{rep}(f(\text{rep}(a))) = \text{rep}(a)$, which implies $r = \text{rep}(a) \in [a]$.
Second statement of condition (5): for any class $[a] \neq [r]$, $\text{rep}(a) \neq r$, and there exists $n \in \mathbb{N}$ such that $(\text{rep} \circ f)^{(n)}(\text{rep}(a)) = r$. 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. The time complexity becomes $\delta(\text{time(Adj)} + \text{time(rep)})$. The memory required is δ times the size of an object.

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

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)

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

- (S, δ, Adj) satisfies the conditions (1) to (3) in Definition 2.1.
- $<$ is a total order on S.
- Only the maximum element r of the total order satisfies $\max_<(\{a \in S : a = \text{Adj}(r, i) \text{ for some } i\} \cup \{r\}) = r$.

Proposition 2.7

A reverse search structure with total order $(S, \delta, \text{Adj}, <)$ together with

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

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

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

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 $(\mathrm{rep} \circ f)^{(n)}(a) = r$. □

2.4 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. Thanks to this property, regular triangulations can be enumerated by applying a vertex enumeration method to the secondary polytope. Refer to [3], [7], [8], [14] and [26] for further information on regular triangulations.

Let $\mathcal{A} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\} \subset \mathbb{R}^k$ 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 $\psi : \mathcal{A} \to \mathrm{I\!R}$ having the following property. For $P = \operatorname{conv}\left\{\binom{\boldsymbol{a}_1}{\psi_1}, \ldots, \binom{\boldsymbol{a}_n}{\psi_n}\right\}$, and π the projection π : $\mathrm{I\!R}^{k+1} \to \mathrm{I\!R}^k$ with $\pi\binom{\boldsymbol{x}}{x_{k+1}} = \boldsymbol{x}$, $T = \{\pi(F) : F \text{ is a lower facet of } P\}$. Here F being a lower facet means, $F = \{\boldsymbol{x} \in P : \boldsymbol{cx} = c_0\}$ is a facet with $\boldsymbol{cx} \leq c_0$ valid for P and $c_{d+1} < 0$.

Let T be a triangulation of $\mathcal{A}$. The *volume vector* for T is a vector $\varphi_T : \mathcal{A} \to \mathrm{I\!R}$ with $\varphi_T(\omega) = \sum_{\sigma \in T : \omega \in \mathrm{vert}(\sigma)} \mathrm{vol}(\sigma)$, 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 $\mathrm{I\!R}^{\mathcal{A}}$ 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 2.10 ([7, 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.

2.5 Enumerating regular triangulations by reverse search

Regular triangulations can be enumerated by reverse search [17]. We restate this result in the context of reverse search structure with total order, which we defined in Definition 2.6 and Theorem 2.7.

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

Definition 2.11 ([17])

The reverse search structure for regular triangulations of an arbitrary point configuration $\mathcal{A}$ is

- $S = \{\text{regular triangulation}\}$
- $\mathrm{Adj}(T, i) =$ (the i-th regular triangulation which can be modified from T along a circuit)
- $f(T) = \begin{cases} \mathrm{Adj}(T,i) & \text{if the largest regular triangulation } \mathrm{Adj}(T,i) \text{ among those adjacent to } T \text{ is larger than } T\text{, in respect of lexicographic order of volume vectors} \\ T & \text{otherwise} \end{cases}$

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

Definition 2.12 (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 $(N^n, N^{n-1}, \dots, N)$ 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.

Theorem 2.13 ([17])

The structure of Definition 2.11 enables reverse search. The time complexity is $O((d+1)s\,\mathrm{LP}(n-d-1,(d+1)s)\#\mathcal{R})$, where s is the upper bound of number of simplices contained in a regular triangulation and $\mathrm{LP}(n,m)$ is the time required to solve a linear programming problem with m strict inequalities constraints in n variables, and $\mathcal{R}$ is the set of regular triangulations. The memory required is several times the size of a triangulation.

2.6 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 2.14 (equivalence on simplices and triangulations)

Let $g \in G$.

- G acts on $\mathcal{A} = \{\boldsymbol{a}_1, \dots, \boldsymbol{a}_n\}$.
- The action of G on a simplex of $\mathcal{A}$ is induced by the action on its vertices: $g\,\mathrm{conv}\{\boldsymbol{a}_{i_1}, \dots, \boldsymbol{a}_{i_m}\} = \mathrm{conv}\{g\boldsymbol{a}_{i_1}, \dots, g\boldsymbol{a}_{i_m}\}$.
- 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, this becomes a reverse search structure for classes with order. Theorem 2.5 leads the following.

Theorem 2.15 (enumerating classes of regular triangulations)

By reverse search structure, total order and equivalence relation defined in Definition 2.11, 2.12 and 2.14, 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 same as the case of Theorem 2.13 and time(rep) in general is as in Lemma 2.16.

Lemma 2.16

The representative of a class is the maximum element. Its time complexity time(rep) 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. □

2.7 Products of two simplices

2.7.1 $\Delta_k \times \Delta_l$

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 2.1 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 2.2.

2.7.2 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.

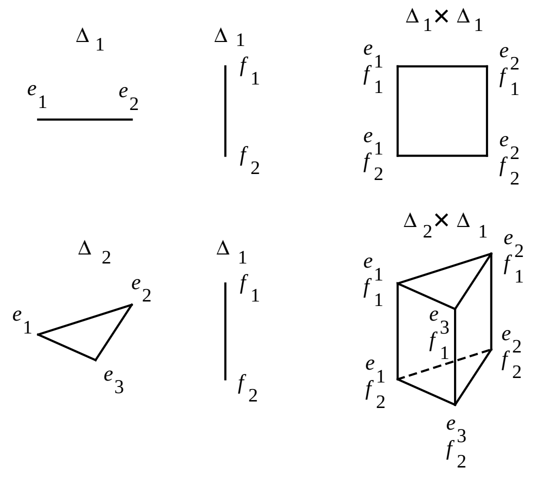

Figure 2.1: Product of simplices: $\Delta_1 \times \Delta_1$ and $\Delta_2 \times \Delta_1$

Definition 2.17

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 2.14. For example, the triangulations T_1 and T_2 in Fig.2.2 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 2.15, 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 by modifying the arguments on complexity shown below.

2.7.3 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 Fig.2.2 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

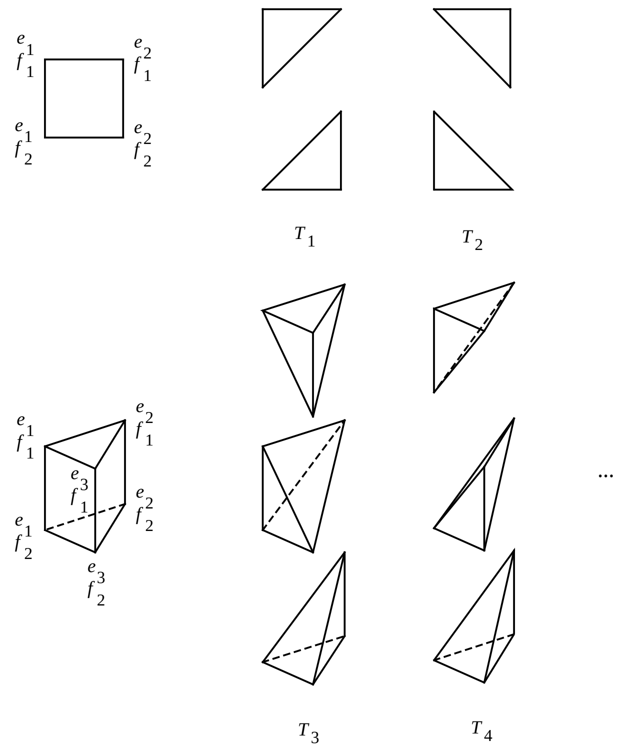

Figure 2.2: Triangulations for $\Delta_1 \times \Delta_1$ and $\Delta_2 \times \Delta_1$

volume vectors 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 2.10).

We introduced a total order on the regular triangulation by lexicographic order of their corresponding volume vectors (Definition 2.12). 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 Fig.2.2, T_2 becomes the representative of the class $\{T_1, T_2\}$.

Lemma 2.18

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 $O((l+1)!\,(k+1)^2(l+1)) = O(l!\,k^2l^2)$. $\square$

This is faster than the time complexity for the general case in Lemma 2.16.

2.7.4 Enumerating classes of regular triangulations

Proposition 2.19

The enumeration algorithm for classes of regular triangulations in Theorem 2.15, 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 $O(l!\,k^2l^2)$ by Lemma 2.18. 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. $\square$

2.8 Hypercubes

2.8.1 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

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

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

2.8.2 The symmetry of C_d

We define the symmetry of C_d as follows.

Definition 2.20

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 $(g_1, \dots, g_d, h) \in S_2 \times \cdots \times S_2 \times S_d$ acts on the vertices of C_d as $(g_1, \dots, g_d, h) \begin{pmatrix} \boldsymbol{e}_{i_1} \\ \vdots \\ \boldsymbol{e}_{i_d} \end{pmatrix} = \begin{pmatrix} \boldsymbol{e}_{g_1(i_{h(1)})} \\ \vdots \\ \boldsymbol{e}_{g_d(i_{h(d)})} \end{pmatrix}$.

This action consists of affine maps defining bijections on C_d and $\operatorname{vert}(C_d)$. Thus it suffices the conditions for the group. Actions and equivalence relations on simplices and triangulations are induced as in Definition 2.14. As shown in Theorem 2.15, 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.

2.8.3 Enumerating classes of regular triangulations

Proposition 2.21

The enumeration algorithm for classes of regular triangulations in Theorem 2.15, 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.

Chapter 3

Polytopes of triangulations

We consider facets of independent set polytopes of intersection graphs of simplices. We deal with two intersection graphs. The intersection graph of d-simplices and the graph of $(d-1)$-simplices for point configurations having dimension d. The independent set polytopes are the convex hulls of the incidence vectors of the independent sets of these graphs.

As a special case, we deal with points spanning the plane. We give a proof that an inequality known to be useful in minimum weight triangulation is defining a facet of the independent set polytope.

3.1 Introduction

We are given n points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ whose convex hull has dimension d, and are interested in their triangulations. All the simplices we consider should have vertices among the given points.

Let us define two simplices to "intersect improperly" if their intersection as point sets is not a face for at least one of them.

We want to consider two intersection graphs and their set of independent sets. The intersection graph of d-simplices has the d-simplices as vertices. The intersecting graph of $(d-1)$-simplices has the $(d-1)$-simplices as vertices. In both graphs, two vertices are connected by an edge if their corresponding simplices are intersecting improperly. An independent set of a graph was a subset of vertices with no edges among them.

The collection of independent sets for these two intersection graphs are set systems with the base set the d-simplices or the $(d-1)$-simplices. The incidence vectors of independent sets become points with $0, 1$ elements in a space with dimension the size of the base set. The convex hull of these incidence vectors becomes a full dimensional $0/1$-polytope.

For any triangulation, the set of d-simplices (or $(d-1)$-simplices) is and independent set in the intersection graph of d-simplices (or $(d-1)$-simplices). Consider the convex hull of the incidence vectors of these sets. For both the d and $(d-1)$-dimensional case, this convex hull also is a 0/1-polytope and is a subset of the convex hull of all the independent sets.

We are interested in the relation between these four polytopes. What kind of facets do they have? What are their dimension? Are the convex hulls of triangulations a face of the convex hulls of independent sets? If so, by what kind of inequalities are their supporting hyperplane given? What kind of relation does the d and $(d-1)$-dimensional case have: what mappings are between them, how will an inequality of the one be reflected to the other?

The case where the given points span a plane is of special interest. Here we consider not only the convex hull of triangulations, but also the convex hull of spanning triangulations. In this case, the number of triangles or edges in a triangulation becomes constant by Euler's formula.

3.2 The general case

3.2.1 Independent set polytope

We are given a simple undirected graph. Let V be the set of vertices. An independent set is a subset $I \subset V$ with no edges among them. The incidence vector of an independent set I is a vector in $\#V$ dimension space with entries one for dimensions corresponding to the elements in I and zeros for others. The *independent set polytope* of the given graph is the polytope made as the convex hull of these incidence vectors. Let P denote this polytope.

The followings are well-known relation between the graph and its independent set polytope.

We are considering simple graphs, so there are no self-loops.

Lemma 3.1

For any $v_i \in V$, $\{v_i\}$ is an independent set. The empty set is also an independent set. Thus all of the unit vectors and the origin in $\mathbb{R}^V$ belong to P. Thus P has full dimension.

Lemma 3.2 (nonnegative constraints)

For any $v_i \in V$, $\{v_i\}$ is an independent set. Let x_i be the corresponding axis. The inequality $x_i \geq 0$ is valid for P and defines its facet.

Lemma 3.3

Let $v_i \in V$ be an isolated vertex (i.e. no edge is adjacent), and x_i the corresponding axis. The independent set polytope P is equal to the product of a line segment $[0, 1]$ (in x_i) and the projection of P to the hyperplane $x_i = 0$ (in other dimensions).

Lemma 3.4 (clique cut)

Let $X \subset V$ be a clique (i.e. complete subgraph) in the given graph. Then the inequality $\sum_{v_i \in X} x_i \leq 1$ is valid for P. It defines a facet of P if and only if the clique X is maximal (in the sense of set inclusion).

Lemma 3.5 (odd-cycle cut)

Let $X \subset V$ be the vertices of an odd-cycle of length $2k+1$ in the given graph. Then the inequality $\sum_{v_i \in X} x_i \leq k$ is valid for P. It defines a facet of P if and only if no point in $V \setminus X$ is adjacent to some vertex in any independent subset of X of size k.

The nonnegative constraints, clique cuts and odd-cycle cuts give faces or facets of the independent set polytope. However, facets of other type exist. Giving descriptions to facets of independent set polytopes is an important topic in Integer Programming [19]. Let us observe what kind of shapes the inequalities, especially there facets, have.

Let $\boldsymbol{a}^T\boldsymbol{x} \leq b$ with $\boldsymbol{a} \in \mathbb{R}^V, b \in \mathbb{R}$ be a valid inequality defining a face of the independent set polytope. For proper faces, $\boldsymbol{a} \neq \boldsymbol{0}$. Since the empty set is an independent set of the graph, $\boldsymbol{0}$ is a vertex of the polytope, so $b \geq 0$.

In case $b = 0$, $\boldsymbol{a} \leq \boldsymbol{0}$. Because, for any $v_i \in V$, $\{v_i\}$ is an independent set, and $b_i \leq 0$ for the incidence vector of this set. These inequalities are implied by the nonnegative constraints of the variables. The corresponding faces are subfaces of the facets defined by these constraints.

In case $b > 0$, $\boldsymbol{a} \geq \boldsymbol{0}$ for facets. The proof is as follows. Since a facet cannot be empty, there should exist a positive component for $\boldsymbol{a}$. Suppose $a_i < 0$ for $v_i \in V$. Let $\boldsymbol{a}'$ be a vector with the i-th component zero and other components same as $\boldsymbol{a}$. The inequality $\boldsymbol{a}'^T\boldsymbol{x} \leq b$ is valid, and the face it defines properly includes the face defined by $\boldsymbol{a}^T\boldsymbol{x} \leq b$. (It has higher dimension in the x_i dimension.) This contradicts our supposition that $\boldsymbol{a}^T\boldsymbol{x} \leq b$ was defining a facet.

And further, since $\{v_i\}$ is independent for any $v_i \in V$, $a_i \leq b$.

As a whole, there are two types of facets. Those for $b = 0$ were the nonnegative constraints $x_i \geq 0$ for all $v_i \in V$. Those for $b > 0$ were $\boldsymbol{a}^T\boldsymbol{x} \leq b$ with $\boldsymbol{a} \geq \boldsymbol{0}$, $\neq \boldsymbol{0}$ and $a_i \leq b$ for any $v_i \in V$.

Facets are faces with dimension $\#V - 1$. The following lemma gives information on the dimension of faces. The support of a vector $\boldsymbol{x}$ is $\mathrm{supp}(\boldsymbol{x}) = \{i : x_i \neq 0\}$.

Lemma 3.6

Let $\boldsymbol{a}^T\boldsymbol{x} \leq b$ with $\boldsymbol{a} \geq \boldsymbol{0}$, $\neq \boldsymbol{0}$, $b > 0$ be an inequality defining a face of the independent set polytope. This inequality is valid for the independent set polytope in $\mathbb{R}^{\mathrm{supp}(\boldsymbol{a})}$ of the subgraph induced by vertices $\mathrm{supp}(\boldsymbol{a}) \subset V$. Let $\#\,\mathrm{supp}(\boldsymbol{a}) - d_1$ be the dimension of the face this inequality defines here. The incidence vectors for some independent sets of this subgraph may give an equality for this inequality. Let d_2 be the number of vertices in $V \setminus \mathrm{supp}(\boldsymbol{a})$ which is adjacent to some vertex for any of these independent sets. The face of the independent set polytope of the original graph defined by this inequality has dimension $\#V - d_1 - d_2$.

So, we obtain a facet when $d_1 = 1$ and $d_2 = 0$.

3.2.2 Independent set system of d-simplices

Let $\mathcal{C}_d$ be the set of d-simplices with vertices among $\{\boldsymbol{p}_1, \dots, \boldsymbol{p}_n\}$. The intersection graph of d-simplices has $\mathcal{C}_d$ as vertices. Two vertices are connected by an edge if the corresponding d-simplices are intersecting improperly. The independent set polytope P_d of this graph is the convex hull of the incidence vectors of the independent sets of this graph. This is an 0/1-polytope in $\#\mathcal{C}_d$ dimensional space.

Let $\boldsymbol{v}$ be a vector in this space with each element the volume of the corresponding d-simplex. The inequality $\boldsymbol{v}^T\boldsymbol{x} \leq \mathrm{vol}(\mathrm{conv}\{\boldsymbol{p}_1, \dots, \boldsymbol{p}_n\})$, where vol is the volume, is valid for P_d. The vertices of the face defined by this inequality are the incidence vectors of d-simplices of the triangulations. This face is known as the universal polytope and studied in [3] [6].

3.2.3 Independent set system of $(d-1)$-simplices

Let $\mathcal{C}_{d-1}$ be the set of $(d-1)$-simplices with vertices among $\{\boldsymbol{p}_1, \dots, \boldsymbol{p}_n\}$. The intersection graph of $(d-1)$-simplices has $\mathcal{C}_{d-1}$ as vertices. Two vertices are connected by an edge if the corresponding $(d-1)$-simplices are intersecting improperly. The independent set polytope P_{d-1} of this graph is the convex hull of the incidence vectors of the independent sets of this graph. This is an 0/1-polytope in $\#\mathcal{C}_{d-1}$ dimensional space.

We define a $(d-1)$-simplex $\sigma \in \mathcal{C}_{d-1}$ to be a *boundary simplex* if it is included in the boundary of the convex hull of the given points $\mathrm{conv}\{\boldsymbol{p}_1, \dots, \boldsymbol{p}_n\}$. Otherwise, it is called an *interior simplex.* Lemma 3.3 leads the following.

Lemma 3.7

Let $\sigma \in \mathcal{C}_{d-1}$ be a boundary $(d-1)$-simplex. Let x_σ be the corresponding axis. Vertex σ is isolated in the intersection graph of $(d-1)$-simplices. Thus P_{d-1} is equal to the product of a line segment $[0, 1]$ (in the axis x_σ) and the projection of P_{d-1} (to the other dimensions).

3.2.4 The relation between the d and $d-1$ dimension case

The boundary of a d-simplex consists of $d+1$ $(d-1)$-simplices. We can define a map from $\mathbb{R}^{\mathcal{C}_d}$ to $\mathbb{R}^{\mathcal{C}_{d-1}}$ by taking the boundary for each simplex.

Take an arbitrary triangulation. For any boundary $(d-1)$-simplex, there are either zero or one d-simplex having this $(d-1)$-simplex in its boundary. For an interior $(d-1)$-simplex, there are either zero or two d-simplex having this $(d-1)$-simplex in its boundary.

It seems appropriate to change the coefficients according to this difference, when taking a linear map from $\mathbb{R}^{\mathcal{C}_d}$ to $\mathbb{R}^{\mathcal{C}_{d-1}}$. We consider the linear map generated by

$$f(\boldsymbol{e}_\sigma) = \sum_{\tau \in \partial\sigma:\ \text{boundary } (d-1)\text{-simplex}} \boldsymbol{e}_\tau + 1/2 \sum_{\upsilon \in \partial\sigma:\ \text{interior } (d-1)\text{-simplex}} \boldsymbol{e}_\upsilon,$$

where $\partial\sigma$ is the set of the facets of a d-simplex σ and $\boldsymbol{e}_\tau \in \mathbb{R}^{\mathcal{C}_d}, \boldsymbol{e}_\sigma, \boldsymbol{ev} \in \mathbb{R}^{\mathcal{C}_{d-1}}$ are unit vectors.

Studying the relation between the polytope $P_d \subset \mathbb{R}^{\mathcal{C}_d}$ and $P_{d-1} \subset \mathbb{R}^{\mathcal{C}_{d-1}}$ is an interesting subject.

3.3 Points spanning the plane

3.3.1 convex n-gon

We are given n points $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ spanning the plane. The dimension is $d = 2$.

We denote an edge with endpoints $\boldsymbol{p}_i$, $\boldsymbol{p}_j$ by e_{ij}, and the set of edges, or $(d-1)$-simplices, by $E = \{e_{ij} : 1 \le i < j \le n\}$. Since there are n points, the number of edges is $\#E = \binom{n}{2}$.

Two edges intersect improperly if their intersection as point set is not a face for at least one of them. The *intersection graph of edges* is the graph with E the vertices and edges between pairs of vertices whose corresponding edges are intersecting improperly. A subset of E is independent if no pair of edges among this subset is intersecting improperly. We denote the collection of independent sets by $\mathcal{I}$. This is an independent set system with base set E. The independent set polytope $P_\mathcal{I}$ is the convex hull of the incidence vectors of the independent sets $\mathcal{I}$.

We denote the collection of sets of edges of triangulations by $\mathcal{T}$. Clearly, $\mathcal{T} \subset \mathcal{I}$. The convex hull of the incidence vectors of $\mathcal{T}$ will be denoted by $P_\mathcal{T}$. Clearly, $P_\mathcal{T} \subset P_\mathcal{I}$.

The polytopes $P_\mathcal{I}$ and $P_\mathcal{T}$ are in $\#E = \binom{n}{2}$ dimensional space. We denote the axis corresponding to edge e_{ij} by x_{ij}.

Since the intersection graph of edges is a simple undirected graph, the properties for the independent set polytope in Subsection 3.2.1 hold.

Lemma 3.8 (restatement of Lemma 3.1)

The empty set and the sets $\{e_{ij}\}$ for any $e_{ij} \in E$ are independent. Thus, the corresponding incidence vectors belong to $P_\mathcal{I}$, and this polytope has full dimension $\#E = \binom{n}{2}$.

Lemma 3.9 (restatement of Lemma 3.2)

For any $e_{ij} \in E$, $x_{ij} \ge 0$ is a valid inequality for $P_\mathcal{I}$, and defines its facet.

We defined two types of $(d-1)$-simplices, the edges. The boundary edges are on the boundary of the convex hull of $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$. The interior edges were the rest.

Lemma 3.10 (restatement of Lemma 3.7)

For any boundary edge e_{ij}, $x_{ij} \leq 1$ is a valid inequality for $P_{\mathcal{I}}$, and defines its facet.

Lemma 3.11

For any interior edge e_{ij}, $x_{ij} \leq 1$ is a valid inequality for $P_{\mathcal{I}}$, and defines its face. The dimension of this face is $\binom{n}{2} - 1 - \#\{e_{kl} \in E : e_{kl}$ intersects improperly with $e_{ij}\}$.

Proof. Lemma 3.6. □

For boundary edges, $e_{ij} \geq 0$ and $e_{ij} \leq 1$ define facets of $P_{\mathcal{I}}$. As in Lemma 3.7, $P_{\mathcal{I}}$ is the product of a polytope in the space corresponding to the boundary edge axis and a polytope in the space of the interior edges. The first component is a hypercube defined by $0 \leq x_{ij} \leq 1$ for boundary edges e_{ij}. The facet inequalities of $P_{\mathcal{I}}$ is exactly the union of facet inequalities for these two components.

Proposition 3.12

Suppose $\{\boldsymbol{p}_{i_1}, \ldots, \boldsymbol{p}_{i_m}\} \subset S$ formed the vertices of a convex m-gon with vertices in this order. Let $E_m = \{e_{i_s i_t} \in E : s - t \neq 0, \pm 1 (\bmod m)\}$ be the interior edges of this m-gon. Let F be the convex hull of the incidence vectors of the independent sets including a set of interior edges of a triangulation of this m-gon. F forms a face of the independent set polytope $P_{\mathcal{I}}$. Its inequality is $\sum_{e_{i_s i_t} \in E_m} x_{i_s i_t} \leq m - 3$. The face F becomes a facet if and only if no edge in $E \setminus E_m$ intersects all sets of $m - 3$ non-intersecting edges in E_m.

Proof. For any triangulation of the convex m-gon, $m - 3$ edges in E_m are used. For any independent set of the whole intersection graph, at most $m - 3$ edges from E_m can be used. So, the inequality $\sum_{e_{i_s i_t} \in E_m} x_{i_s i_t} \leq m - 3$ is valid on $P_{\mathcal{I}}$. The equality holds for independent sets including a set of interior edges of a triangulation of this m-gon. Thus, $F = P_{\mathcal{I}} \cap \{\boldsymbol{x} : \sum_{e_{i_s i_t} \in E_m} x_{i_s i_t} = m - 3\}$, and F is a face of $P_{\mathcal{I}}$.

Let $\mathcal{T}$ be the set of $m - 3$ non-intersecting edges in E_m. In other words, this is the set of interior edges of triangulations of the convex m-gon. Let U be the incidence vectors of $\mathcal{T}$. We use the same indexing for U and $\mathcal{T}$, for example $\boldsymbol{u}_p \in U$ corresponds to $T_p \in \mathcal{T}$. Clearly, the points in U lie on F. Let $\boldsymbol{c} = (1/\#U) \sum_{\boldsymbol{u} \in U} \boldsymbol{u}$ be the centroid of U. This point is included in the convex hull of U, which is a subset of F.

Suppose two edges $e_{i_s i_t}, e_{i_u i_v} \in E_m$ intersect. There exists a pair of triangulations $T_p, T_q \in \mathcal{T}$ such that $T_p \setminus \{e_{i_s i_t}\} \cup \{e_{i_u i_v}\} = T_q$. Let $\boldsymbol{f}_{i_s i_t}$ denote the unit vector with $x_{i_s i_t} = 1$ and other elements zero. In our situation, $\boldsymbol{u}_q - \boldsymbol{u}_p = \boldsymbol{f}_{i_u i_v} - \boldsymbol{f}_{i_s i_t}$. Consider moving $\boldsymbol{c}$ slightly along the orientation $\boldsymbol{f}_{i_u i_v} - \boldsymbol{f}_{i_s i_t}$: $\boldsymbol{c} + (1/\#U)(\boldsymbol{f}_{i_u i_v} - \boldsymbol{f}_{i_s i_t}) = (2/\#U)\boldsymbol{u}_q + \sum_{T_r \in \mathcal{T} \setminus \{T_p, T_q\}} (1/\#U)\boldsymbol{u}_r$. The right-hand side indicates that this point is still in the convex hull of U.

For edges in E_m, the edge $e_{i_1 i_3}$ intersects edges $e_{i_2 i_t}$. The edge $e_{i_2 i_4}$ intersects edges $e_{i_3 i_t}$. Edges $e_{i_s, i_{s+2}}$ intersects edges $e_{i_{s+1} i_t}$. Thus any edge in E_m is an intersecting edge of an intersecting edge of ... an intersecting edge of the edge $e_{i_1 i_3}$. Hence, for any edge $e_{i_s i_t} \in E_m$, moving $\boldsymbol{c}$ slightly along the orientation $\boldsymbol{f}_{i_s i_t} - \boldsymbol{f}_{i_1 i_3}$ leaves the point inside the convex hull of U. This shows that the convex hull of U around $\boldsymbol{c}$ has dimension $\#E_m - 1$ in the directions $\{\boldsymbol{f}_{i_s i_t} : e_{i_s i_t} \in E_m\}$.

The face F becomes a facet if and only if it has full dimension in the other directions, which are $\{\boldsymbol{f}_{ij} : e_{ij} \in E \setminus E_m\}$, We show that this happens if and only if no edge in $E \setminus E_m$ intersects all sets of $m-3$ non-intersecting edges in E_m. Suppose no such edge existed. Then, for any edge $e_{ij} \in E \setminus E_m$, there exists a set of $m-3$ non-intersecting edges in E_m. Let this be $T_p \in \mathcal{T}$. The incidence vector of T_p was $\boldsymbol{u}_p \in U$, and points in U were on F. The point $\boldsymbol{u}_p + \boldsymbol{f}_{ij}$ is also on F. Thus, $\boldsymbol{c} + (1/\#U)\boldsymbol{f}_{ij} = \sum_{T_r \in \mathcal{T} \setminus \{T_p\}} (1/\#U)\boldsymbol{u}_r + (1/\#U)(\boldsymbol{u}_p + \boldsymbol{f}_{ij})$ is on F. This is the point obtained by moving $\boldsymbol{c}$ slightly along $\boldsymbol{f}_{ij}$. Since the choice of $e_{ij} \in E \setminus E_m$ was arbitrary, and F was convex, this is equivalent to F around $\boldsymbol{c}$ having full dimension of $\{\boldsymbol{f}_{ij} : e_{ij} \in E \setminus E_m\}$. □

Corollary 3.13

Let $S = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ be vertices of a convex n-gon. A face of the convex k-gon as in the proposition above forms a facet if and only if the k points are taken consecutively from the n-gon.

This facet is known to be powerful for the minimum weight triangulation problem. It was called geometric cut in [11] and convex polygon cut in [12].

Proposition 3.14

Let $S = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ be vertices of a convex n-gon. The inequalities $x_{ij} \le 1$ (e_{ij}: boundary) and $\sum_{e_{ij}:\text{interior}} x_{ij} \le n-3$ define facets of $P_{\mathcal{I}}$, and

$$P_{\mathcal{T}} = P_{\mathcal{I}} \cap \bigcap_{e_{ij}:\text{boundary}} \{x_{ij} = 1\} \cap \{ \sum_{e_{ij}:\text{interior}} x_{ij} = n-3\}.$$

The dimension of $P_{\mathcal{T}}$ is $\binom{n}{2} - n - 1$.

Proof. The inequalities define facets by Lemma 3.10 and Proposition 3.12. By Euler's formula, triangulations are the independent sets using all of the n boundary edges and $n-3$ edges among the interior edges. For the dimension, remember that $P_{\mathcal{I}}$ could be seen as a product of n-cube corresponding to the boundary-edge-dimensions and a polytope for the interior-edge-dimensions. Similarly, $P_{\mathcal{T}}$ can be seen as a product of a point with all components equal to 1 for the former dimensions and a facet of the polytope in the latter dimensions. □

Appendix A

Programs and examples

Programs for the algorithms and and examples of computations for point configurations in Chapter 1 are given. The language used is *Mathematica* [25]. These materials are available at `http://naomi.is.s.u-tokyo.ac.jp/~fumi/`

A.1 Algorithm 1.6: Enumeration of maximal independent sets of graphs

```
(* mis.nb *)

(*
enumerate

enumerate the triangulations (TRI) and maximal independent
sets (MIS)

*** defined outside ***
NumberOfSimplices = total number of maximal dimensional
                    simplices
IntersectProperQ[i,j] = True, False
                        whether simplex[i] and simplex[j]
                        intersect properly or not
OutputFile = file to append output
simplexVolume[i] = volume of the i-th simplex
wholeVolume = volume of the whole convex hull

*** variables ***
current = the current set
level = 0, 1, ..., NumberOfSimplices
trace = array of the trace of search
        index: 1, ..., NumberOfSimplices
        contents: 0 = not visited
                  1 = Union[currentset,level] was
                      independent. visiting that unique
```

```
                        child
                    2 = Union[currentset,level] was
                          dependent. visiting the first
                          child "currentset"
                    3 = Union[currentset,level] was
                          dependent. visiting the second
                          child, which is the maximal subset
                          of Union[currentset,level]
                          including "level"
candidate = candidate of the second child
*)
ClearAll[enumerate];
enumerate:=
Module[
    {current={}, level = 1,
     trace,
     candidate},
    Do[trace[i]=0, {i, NumberOfSimplices}];
    While[
        level > 0,
        If[ level <= NumberOfSimplices,
            (* internal node *)
            Switch[ ((***
                     Print["trace[",level,"] is ",
                         trace[level]];
                     ***)
                     trace[level]),
                (* 0 = visiting the unvisited child *)
                0,
                If[ (* Union[current,level] independent ? *)
                    SetIntersectProperQ[current,level],
                    (AppendTo[current,level];
                     trace[level] = 1; level++),
                    ((* no change to current *)
                     trace[level] = 2; level++)]
                (***
                ; Print["node 0, level ",level,", current ",
                    current]
                ***)
                ,
                (* 1 = ascending from the unique child *)
                1,
                (current = Complement[current,{level}];
                 trace[level] = 0; level--)
                (***
                ; Print["node 1, level ",level,", current ",
                    current]
                ***)
```

```
,
(* 2 = ascending from the first child *)
2,
If[ (* the second child exist ? *)
    And[((* making candidate *)
         candidate = {};
         Scan[
            (If[IntersectProperQ[#,level],
                AppendTo[candidate,#]])&,
            current];
         AppendTo[candidate,level]
         (***
         ; Print["candidate ",candidate]
         ***)
         );
        (* checking the maximality of
            candidate in {1,...,level} *)
        MaximalQ[candidate,level],
        (* is parent the lexicographically
            smallest parent of candidate ? *)
        LexMinParentQ[current,
            candidate,
            level]],
    (* the second child exists *)
    (current = candidate;
     trace[level] = 3; level++),
    (* the second child does not exist *)
    ((* no change to current *)
     trace[level] = 0; level--)]
(***
; Print["node 2, level ",level,", current ",
    current]
***)
,
(* 3 = ascending from the second child *)
3,
(current = Complement[current,{level}];
 Do[
     If[ And[
             Not[MemberQ[current,i]],
             SetIntersectProperQ[current,i]],
    current=Union[current,{i}]],
    {i,level-1}];
 trace[level] = 0; level--)
 (***
 ; Print["node 3, level ",level,", current ",
     current]
 ***)
```

```
                ],
              (* bottom *)
              (If[
                  Apply[
                      Plus,
                      Abs[
                          Map[
                              simplexVolume,
                              current]]]
                  ==wholeVolume,
                  ((*Print["TRI ",current];*)
                   PutAppend[{TRI,current},OutputFile]),
                  ((*Print["MIS ",current];*)
                   PutAppend[{MIS,current},OutputFile])];
               level--)]]]

(*
SetIntersectProperQ[list,j] = True, False
    whether all simplex[i] for some element i of list
    intersects properly with simplex[j] or not
*)
ClearAll[SetIntersectProperQ];
SetIntersectProperQ[list_,j_] :=
    If[ list=={},
        True,
        If[ IntersectProperQ[Part[list,1],j],
            SetIntersectProperQ[Rest[list],j],
            False]]

(*
MaximalQ[set,level] = True, False
    whether set is a maximal independent set in {1,...,level}
    or not
*)
ClearAll[MaximalQ];
MaximalQ[set_,level_] :=
    Module[{i=1,j,l=Length[set]},
        While[
            And[i<=level,
                Or[ MemberQ[set,i],
                    (j=1;
                     While[
                         And[j<=l,
                             IntersectProperQ[i,set[[j]]]],
                        j++];
                     j<=l)]],
            i++];
        (***
        If[ i>level,
            Print[set," is maximal in {1,...,",level,"}"],
            Print[set," is not maximal in {1,...",level,"}"]];
        ***)
        i>level]

(*
LexMinParentQ[parent,child,level] = True, False
    whether parent is the lexicographically smallest parent of
    candidate or not
*)
ClearAll[LexMinParentQ];
LexMinParentQ[parent_,child_,level_] :=
    Module[{temporaryset,i=1,m},
        m = If[ parent=={},
                level-1,
                Last[parent]];
        temporaryset = Complement[child,{level}];
        While[MemberQ[parent,i],
            (AppendTo[temporaryset,i];
             i++)];
        While[
            And[i < m,
                Not[SetIntersectProperQ[temporaryset,i]]],
            (i++;
             While[MemberQ[parent,i],
                (AppendTo[temporaryset,i];
                 i++)])];
        (***
        If[ i >= m,
            Print["parent is lex min parent"],
            Print["parent is not lex min parent"]];
        ***)
        i >= m]
```

A.2 Naive enumeration of *d*-simplices (Section 1.4)

```
(* generalpoints.nb *)

enumerateSimplices :=

(

(*
simplex[i] = list of the indices of its vertices
*)
ClearAll[simplex,simplexVolume,NumberOfSimplices];
Module[{i=1,
        det,
        c=1/(Length[vertex[1]]!),
        l=Length[vertex[1]]+1},
    Scan[
        (If[
            (det =
                Det[
                    Append[
                        Transpose[
                            Map[
                                vertex,
                                #]],
                        Table[1,{l}]]];
             det!=0),
            (simplex[i]=#;
             simplexVolume[i]=c det;
             ++i)])&,
        KSubsets[Range[NumberOfVertices],l]];
    NumberOfSimplices = i-1];

)
```

A.3 Algorithm 1.9: Testing intersection of *d*-simplices

```
(*
IntersectProperQ[i,j] = True, False
                        whether simplex[i] and simplex[j]
                        intersect properly or not *)
ClearAll[IntersectProperQ];
IntersectProperQ[i_,j_]:=
    Module[
        {OnlyInI = Complement[
                    simplex[i],
                    simplex[j]],
         OnlyInJ = Complement[
                    simplex[j],
                    simplex[i]],
         BothInIJ = Intersection[simplex[i],simplex[j]],
         FirstInBothIJ,
         Difference,
         matrix,
         lp},
         Difference = Length[OnlyInI];
         If[
            BothInIJ=={},
            (matrix =
                Map[(vertex[#]-vertex[simplex[i][[1]]])&,
                    simplex[j]] .
                Inverse[
                    Map[(vertex[#]-vertex[simplex[i][[1]]])&,
                        Drop[simplex[i],1]]];
             matrix =
                Transpose[matrix];
             lp = LinearProgramming[
                Join[{1},Table[0,{Difference-1}]],
                Join[matrix,
                    {Table[1,{Difference}]},
                    {Table[-1,{Difference}]},
                    {-Apply[Plus,matrix]}
                    ],
                Join[Table[0,{Difference-1}],{1,-1,-1}]]),
            (FirstInBothIJ = BothInIJ[[1]];
             BothInIJ = Drop[BothInIJ,1];
             matrix =
                Map[(vertex[#]-vertex[FirstInBothIJ])&,
                    Join[OnlyInJ,BothInIJ]] .
                Inverse[
                    Map[(vertex[#]-vertex[FirstInBothIJ])&,
                        Join[OnlyInI,BothInIJ]]];
             matrix =
        Take[
            Transpose[
                Take[
                    matrix,
                    Difference]],
            Difference];
     lp = LinearProgramming[
        Join[{1},Table[0,{Difference-1}]],
        Join[matrix,
            {Table[1,{Difference}]}
            (*,
            {Table[-1,{Difference}]}
            *)
            ],
        Join[Table[0,{Difference}],{1(*,-1*)}]])];
Switch[
   Head[lp],
   LinearProgramming,
   True,
   List,
   False,
   _,
   Error]

   (*Print[
   Join[{1},Table[0,{Difference-1}]],
   Join[matrix,
       {Table[1,{Difference}]},
       {Table[-1,{Difference}]}],
   Join[Table[0,{Difference}],{1,-1}]];*)]
```

A.4 Memorizing the intersection graph (Section 1.5)

```
memorizeIntersectProperQ :=

(

ClearAll[tempTable,IntersectProperQTable];
Print[
Timing[
    tempTable=
        Table[
            If[ i>j,
                IntersectProperQ[i,j],
                Null],
            {i,NumberOfSimplices},{j,NumberOfSimplices}];]
];

Print[
Timing[
    IntersectProperQTable=
        Table[
            If[ i>j,
                tempTable[[i,j]],
                If[ i<j,
                    tempTable[[j,i]],
                    Null]],
            {i,NumberOfSimplices},{j,NumberOfSimplices}];]
];

ClearAll[tempTable];

ClearAll[IntersectProperQ];
IntersectProperQ[i_,j_] :=
    IntersectProperQTable[[i,j]]

)
```

A.5 A nonregular triangulation

As in Theorem 1.8, Algorithm 1.6 enumerates all triangulations including nonregular ones. The nonregular triangulation `{TRI, {3, 5, 8, 13, 16, 18, 20}}` in the figure below is enumerated.

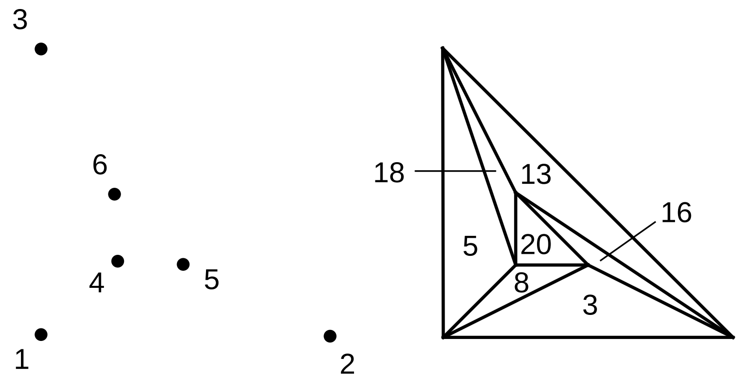

Figure A.1: A nonregular triangulation

```
(* Nonregular.nb *)

<<DiscreteMath‘Combinatorica‘

OutputFile = "/home/asuka0/fumi/tri/Nonregular.output";

<<"/home/asuka0/fumi/tri/mis.m";
<<"/home/asuka0/fumi/tri/generalpoints.m";

ClearAll[vertex,NumberOfVertices];
vertex[1]={ 0, 0};
vertex[2]={ 4, 0};
vertex[3]={ 0, 4};
vertex[4]={ 1, 1};
vertex[5]={ 2, 1};
vertex[6]={ 1, 2};
NumberOfVertices=6;

enumerateSimplices;

ClearAll[wholeVolume];
wholeVolume = 8;

Timing[enumerate]

    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    General::"stop":
        "Further output of \!\(ConstrainedMin :: \"nsat\"\) will be suppressed \
         during this calculation."
    {26.8 Second,Null}

??simplex

    "Global`simplex"
    simplex[1] = {1, 2, 3}
    simplex[2] = {1, 2, 4}
    simplex[3] = {1, 2, 5}
    simplex[4] = {1, 2, 6}
    simplex[5] = {1, 3, 4}
    simplex[6] = {1, 3, 5}
    simplex[7] = {1, 3, 6}
    simplex[8] = {1, 4, 5}
    simplex[9] = {1, 4, 6}
    simplex[10] = {1, 5, 6}
    simplex[11] = {2, 3, 4}
    simplex[12] = {2, 3, 5}
    simplex[13] = {2, 3, 6}
    simplex[14] = {2, 4, 5}
    simplex[15] = {2, 4, 6}
    simplex[16] = {2, 5, 6}
    simplex[17] = {3, 4, 5}
    simplex[18] = {3, 4, 6}
    simplex[19] = {3, 5, 6}
    simplex[20] = {4, 5, 6}

??simplexVolume

    "Global`simplexVolume"
    simplexVolume[1] = 8
    simplexVolume[2] = 2
    simplexVolume[3] = 2
    simplexVolume[4] = 4
    simplexVolume[5] = -2
    simplexVolume[6] = -4
    simplexVolume[7] = -2
    simplexVolume[8] = -1/2
    simplexVolume[9] = 1/2
    simplexVolume[10] = 3/2
    simplexVolume[11] = 4
    simplexVolume[12] = 2
    simplexVolume[13] = 2
    simplexVolume[14] = -1/2
    simplexVolume[15] = -3/2
    simplexVolume[16] = -1/2
    simplexVolume[17] = 3/2
    simplexVolume[18] = 1/2
    simplexVolume[19] = -1/2
    simplexVolume[20] = 1/2

!!"/home/asuka0/fumi/tri/Nonregular.output"

    {TRI, {1}}
    {TRI, {4, 7, 13}}
    {TRI, {3, 5, 8, 12, 17}}
    {TRI, {3, 5, 8, 12, 18, 19, 20}}
    {TRI, {3, 5, 8, 13, 16, 18, 20}}
    {TRI, {3, 7, 8, 9, 12, 19, 20}}
    {TRI, {3, 7, 8, 9, 13, 16, 20}}
    {TRI, {3, 7, 10, 12, 19}}
    {TRI, {3, 7, 10, 13, 16}}
    {TRI, {3, 6, 12}}
    {TRI, {2, 5, 11}}
    {TRI, {2, 5, 13, 14, 16, 18, 20}}
    {TRI, {2, 5, 13, 15, 18}}
    {TRI, {2, 5, 12, 14, 17}}
    {TRI, {2, 5, 12, 14, 18, 19, 20}}
    {TRI, {2, 7, 9, 12, 14, 19, 20}}
    {TRI, {2, 7, 9, 13, 14, 16, 20}}
    {TRI, {2, 7, 9, 13, 15}}
```

A.6 Schönhardt's polyhedron

Algorithm 1.6 enumerates, not only triangulations, but also maximal independent sets which are not triangulations. Schönhardt's polyhedron mentioned in Section 1.2 was such example. The three outer tetrahedra in the right of the figure below corresponds to the maximal independent set `{MIS, {4, 8, 13}}`.

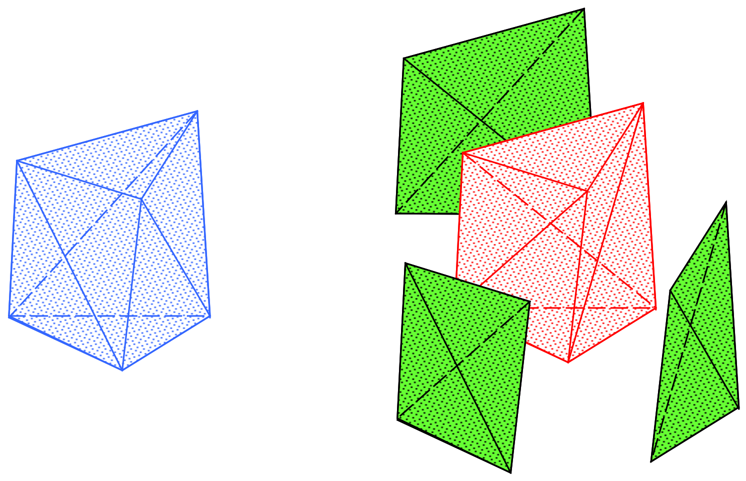

Figure A.2: A maximal independent set which is not a triangulation

```
(* Schonhardt.nb *)

<<DiscreteMath`Combinatorica`

OutputFile = "/home/asuka0/fumi/tri/Schonhardt.output";

<<"/home/asuka0/fumi/tri/mis.m";
<<"/home/asuka0/fumi/tri/generalpoints.m";

ClearAll[vertex,NumberOfVertices];
vertex[1]={    0,    0,    0};
vertex[2]={    1,    0,    0};
vertex[3]={    0,    1,    0};
vertex[4]={  1/8, -1/8,    1};
vertex[5]={    1,  1/8,    1};
vertex[6]={ -1/8,    1,    1};
NumberOfVertices=6;

enumerateSimplices;

ClearAll[wholeVolume];
wholeVolume = 73/128;
```

```
Timing[enumerate]

    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    General::"stop":
         "Further output of \!\(ConstrainedMin :: \"nsat\"\) will be suppressed \
          during this calculation."
    {5.44 Second,Null}

??simplex

    "Global`simplex"
    simplex[1] = {1, 2, 3, 4}
    simplex[2] = {1, 2, 3, 5}
    simplex[3] = {1, 2, 3, 6}
    simplex[4] = {1, 2, 4, 5}
    simplex[5] = {1, 2, 4, 6}
    simplex[6] = {1, 2, 5, 6}
    simplex[7] = {1, 3, 4, 5}
    simplex[8] = {1, 3, 4, 6}
    simplex[9] = {1, 3, 5, 6}
    simplex[10] = {1, 4, 5, 6}
    simplex[11] = {2, 3, 4, 5}
    simplex[12] = {2, 3, 4, 6}
    simplex[13] = {2, 3, 5, 6}
    simplex[14] = {2, 4, 5, 6}
    simplex[15] = {3, 4, 5, 6}

??simplexVolume

    "Global`simplexVolume"
    simplexVolume[1] = -1/6
    simplexVolume[2] = -1/6
    simplexVolume[3] = -1/6
    simplexVolume[4] = 1/24
    simplexVolume[5] = 3/16
    simplexVolume[6] = 7/48
    simplexVolume[7] = -7/48
    simplexVolume[8] = 1/24
    simplexVolume[9] = 3/16
    simplexVolume[10] = -67/384
    simplexVolume[11] = -3/16
    simplexVolume[12] = -7/48
    simplexVolume[13] = 1/24
    simplexVolume[14] = -67/384
    simplexVolume[15] = -67/384
```

```
!!"/home/asuka0/fumi/tri/Schonhardt.output"

    {TRI, {1, 8, 11, 15}}
    {TRI, {1, 8, 12, 13, 14}}
    {TRI, {3, 4, 6, 10, 13}}
    {TRI, {3, 5, 13, 14}}
    {TRI, {2, 4, 7, 8, 15}}
    {MIS, {4, 8, 13}}
    {TRI, {2, 4, 9, 10}}
```

A.7 Products of two simplices

A.7.1 Enumeration of vertices

```
(* deltakdeltal.nb *)

(*
products of two simplices $\Delta_k \times \Delta_l$
k and l are the dimensions
*)
enumerateVerticesDeltakDeltal :=
(

(*
vertexmatrix is
the matrix of the full dimensional point arrangement
*)
ClearAll[vertexmatrix];
vertexmatrix=
Transpose[
    Drop[
        Drop[
            Transpose[
                Flatten[
                    Outer[
                        Join,
                        IdentityMatrix[k+1],
                        IdentityMatrix[l+1],
                        1],
                    1]],
            {k+1}],
        {k+l+1}]];

(*
vertex[i] = USED vertices of simplex i
the full dimensional point arrangement of (k+1)(l+1)
vertices
*)
ClearAll[vertex];
Do[
    vertex[i] = vertexmatrix[[i]],
    {i,(k+1)(l+1)}];

ClearAll[vertexmatrix]

)
```

A.7.2 $\Delta_2 \times \Delta_1$

The polytope $\Delta_2 \times \Delta_1$ and its triangulations are shown in Figure 1.1 and 1.2.

```
(* delta2delta1.nb *)

(*
products of two simplices $\Delta_k \times \Delta_l$
k and l are the dimensions
*)

ClearAll[k,l];
k=2;
l=1;

OutputFile = "/home/asuka0/fumi/tri/delta2delta1.output";

<<DiscreteMath`Combinatorica`

<<"/home/asuka0/fumi/tri/mis.m";
<<"/home/asuka0/fumi/tri/generalpoints.m";
<<"/home/asuka0/fumi/tri/deltakdeltal.m";

enumerateVerticesDeltakDeltal;

ClearAll[NumberOfVertices];
NumberOfVertices = (k+1)(l+1);

enumerateSimplices;

ClearAll[wholeVolume];
wholeVolume = 1/(k! l!);

Timing[enumerate]

    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    General::"stop":
        "Further output of \!\(ConstrainedMin :: \"nsat\"\) will be suppressed \
         during this calculation."
    {3.13 Second,Null}

??vertex

    "Global`vertex"
    vertex[1] = {1, 0, 1}
    vertex[2] = {1, 0, 0}
    vertex[3] = {0, 1, 1}
    vertex[4] = {0, 1, 0}
    vertex[5] = {0, 0, 1}
    vertex[6] = {0, 0, 0}

??simplex

    "Global`simplex"
    simplex[1] = {1, 2, 3, 5}
    simplex[2] = {1, 2, 3, 6}
    simplex[3] = {1, 2, 4, 5}
    simplex[4] = {1, 2, 4, 6}
    simplex[5] = {1, 3, 4, 5}
    simplex[6] = {1, 3, 4, 6}
    simplex[7] = {1, 3, 5, 6}
    simplex[8] = {1, 4, 5, 6}
    simplex[9] = {2, 3, 4, 5}
    simplex[10] = {2, 3, 4, 6}
    simplex[11] = {2, 3, 5, 6}
    simplex[12] = {2, 4, 5, 6}

??simplexVolume

    "Global`simplexVolume"
    simplexVolume[1] = 1/6
    simplexVolume[2] = 1/6
    simplexVolume[3] = 1/6
    simplexVolume[4] = 1/6
    simplexVolume[5] = -1/6
    simplexVolume[6] = -1/6
    simplexVolume[7] = 1/6
    simplexVolume[8] = 1/6
    simplexVolume[9] = -1/6
    simplexVolume[10] = -1/6
    simplexVolume[11] = 1/6
    simplexVolume[12] = 1/6

!! "/home/asuka0/fumi/tri/delta2delta1.output"

    {TRI, {1, 9, 12}}
    {TRI, {1, 10, 11}}
    {TRI, {4, 5, 8}}
    {TRI, {4, 6, 7}}
    {TRI, {3, 5, 12}}
    {TRI, {2, 7, 10}}
```

A.7.3 $\Delta_2 \times \Delta_2$

```
(* delta2delta2.nb *)

(*
products of two simplices $\Delta_k \times \Delta_l$
k and l are the dimensions
*)

ClearAll[k,l];
k=2;
l=2;

OutputFile = "/home/asuka0/fumi/tri/delta2delta2.output";

<<DiscreteMath`Combinatorica`

<<"/home/asuka0/fumi/tri/mis.m";
<<"/home/asuka0/fumi/tri/generalpoints.m";
<<"/home/asuka0/fumi/tri/deltakdeltal.m";

enumerateVerticesDeltakDeltal;

ClearAll[NumberOfVertices];
NumberOfVertices = (k+1)(l+1);

enumerateSimplices;

ClearAll[wholeVolume];
wholeVolume = 1/(k! l!);

Timing[enumerate]

    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    General::"stop":
        "Further output of \!\(ConstrainedMin :: \"nsat\"\) will be suppressed \
         during this calculation."
    {1214.61 Second,Null}
```

A.7.4 $\Delta_2 \times \Delta_2$ (memorizing the intersection graph)

The computation becomes faster by memorizing the intersection graph (Section 1.5). The previous example took 1200 seconds, whereas this one $50 + 100 = 150$ seconds.

```
(*
products of two simplices $\Delta_k \times \Delta_l$
k and l are the dimensions
*)

ClearAll[k,l];
k=2;
l=2;

OutputFile = "/home/asuka0/fumi/tri/delta2delta2.output2";

<<DiscreteMath`Combinatorica`

<<"/home/asuka0/fumi/tri/mis.m";
<<"/home/asuka0/fumi/tri/generalpoints.m";
<<"/home/asuka0/fumi/tri/deltakdeltal.m";

enumerateVerticesDeltakDeltal;

ClearAll[NumberOfVertices];
NumberOfVertices = (k+1)(l+1);

enumerateSimplices;

ClearAll[wholeVolume];
wholeVolume = 1/(k! l!);

memorizeIntersectProperQ;

Timing[enumerate]

    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    ConstrainedMin::"nsat": "The specified constraints cannot be satisfied."
    General::"stop":
        "Further output of \!\(ConstrainedMin :: \"nsat\"\) will be suppressed \
         during this calculation."
    {43.74 Second,Null}
    {0.58 Second,Null}
    {98.95 Second,Null}
```

References

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

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

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

[4] Jesús A. de Loera. Computing regular triangulations of point configurations. Preprint, 1994.

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

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

[7] Israel M. Gelfand, Mikhail M. Kapranov, and Andrei V. Zelevinsky. *Discriminants, Resultants and Multidimensional Determinants.* Birkhäuser, Boston, 1994.

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

[9] Hiroshi Imai and Keiko Imai. Triangulation and convex polytopes. In *Geometry of Toric Varieties and Convex Polytopes*, volume 934 of *RIMS Kokyuroku*, pages 149–166. Research Institute for Mathematical Sciences, Kyoto University, 1996. (in Japanese).

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

[11] Yoshiaki Kyoda. A study of generating minimum weight triangulation within practical time, March 1996.

[12] Yoshiaki Kyoda, Keiko Imai, Fumihiko Takeuchi, and Akira Tajima. A branch-and-cut approach for minimum weight triangulation. In Hon Wai Leong and Hiroshi Imai, editors, *Algorithms and Computation—8th International Symposium, ISAAC '97*, volume 1350 of *Lecture Notes in Computer Science*, pages 384–393, Singapore, 1997. Springer-Verlag, Berlin.

[13] 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.*, 9:558–565, 1980.

[14] Carl W. Lee. Regular triangulations of convex polytopes. In Peter Gritzmann and Bernd Sturmfels, editors, *Applied Geometry and Discrete Mathematics—The Victor Klee Festschrift*, volume 4 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 443–456. American Mathematical Society, 1991.

[15] Carl W. Lee. Subdivisions and triangulations of polytopes. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 14, pages 271–290. CRC Press LLC, Boca Raton, 1997.

[16] Tomonari Masada. An algorithm for the enumeration of regular triangulations, March 1995.

[17] Tomonari Masada, Hiroshi Imai, and Keiko Imai. Enumeration of regular triangulations. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, pages 224–233, New York, 1996. Association for Computing Machinery (ACM).

[18] Joseph O'Rourke. *Art Gallery Theorems and Algorithms*, volume 3 of *International Series of Monographs on Computer Science.* Oxford University Press, New York, 1987.

[19] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Chichester, 1986.

[20] Akiyoshi Shioura, Akihisa Tamura, and Takeaki Uno. An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM J. Comp.*, 26:678–692, 1997.

[21] Bernd Sturmfels. Gröbner bases of toric varieties. *Tôhoku Math. J.*, 43:249–261, 1991.

[22] Bernd Sturmfels. *Gröbner Bases and Convex Polytopes*, volume 8 of *University Lecture Series.* American Mathematical Society, 1996.

[23] Fumihiko Takeuchi and Hiroshi Imai. Enumerating triangulations for products of two simplices and for arbitrary configurations of points. In Tao Jiang and D. T.

Lee, editors, *Computing and combinatorics: third annual international conference; proceedings / COCOON'97*, volume 1276 of *Lecture Notes in Computer Science*, pages 470–481, Shanghai, 1997. Springer-Verlag, Berlin.

[24] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM J. Comput.*, 6:505–517, 1977.

[25] Stephen Wolfram. *The Mathematica Book.* Wolfram Media/Cambridge University Press, Champaign, 3rd edition, 1996.

[26] Günter M. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics.* Springer-Verlag, New York, 1995.

Index