Inference the result of polytope operation in Python

Department of mathematics, JuWon Jang, Giryeon Rark

Contents

1 Introduction

```
polytope > $p=new Polytope<Rational>;

polytope > $p->POINTS=<<".";
polytope (2)> 1 0 0 0
polytope (3)> 1 0 1 0
polytope (4)> .

polytope > print $p->F_VECTOR;
polymake: used package cddlib
 Implementation of the double description method of Motzkin et al.
 Copyright by Komei Fukuda.
 http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html

2
polytope > $q=new Polytope<Rational>;

polytope > $q->POINTS=<<".";
polytope (2)> 1 0 0 0
polytope (3)> 1 1 0 0
polytope (4)> 1 1 1 1
polytope (5)> .

polytope > print $q->F_VECTOR;
3 3
polytope > $m = minkowski_sum($p,$q);

polytope > print $m->F_VECTOR;
6 9 5
```

Figure 1: Minkowski sum in different plane

We live in is three-dimensional, so when the word four-dimensional cannot feel close to us. So we start with operations between polygons and expand dimensions through operations between polygons and polyhedrons, operations between polyhedrons and polyhedrons, and use a program called polymake to create -1,0, 1, 2, ..., examine the number of n-dimensional elements, determine the features of high-dimensional figures to the number of components they have, which make the high-dimensional figures feel friendly to the viewer of this paper However, the number of geometric components that came out of the operation did not allow their features to be determined, and instead sought to explore geometric features. Therefore, it is important to predict the result of the operation used to extend the dimension, and to research the geometric features of the figure that came out through the operation through the expansion of the operation to 4 dimensions To start.

2 Previous Research

2.1 Vector space

2.1.1 Definition of Vector space

$$R^d = \{(x_1, x_2, ..., x_d) | x_1, x_2, ..., x_d \in R\}$$

It is definition of dimension when the number of vector x which enter in the R is d. For example when the 3-dimension formula has been

$$R^3 = \{(x, y, z) | x, y, z \in R\}$$

Each element can expressed by

$$(x, y, z) = x(1, 0, 0) + y(0, 1, 0) + z(0, 0, 1) = x\hat{i} + y\hat{j} + z\hat{k}$$

2.1.2 Dual Vector space

Show vector space after express equation by matrix. At first column vector which means dot is

$$R^d = \{ \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_d \end{bmatrix} : x_1, x_2, ..., x_d \in R\}$$

and row vector which means solute is

$$(R^d)^* = \{[x_1, x_2, ..., x_d] : x_1, x_2, ..., x_d \in R\}$$

For example, 2-dimensional line $ax + by = c$ to $\begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

3-dimensional plane $ax + by + cz = d$ to $\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

additionally matrix e which show i_{th} element is expressed

by $e_i = \begin{bmatrix} 0 \\ 0 \\ ... \\ 0 \\ 1 \\ 0 \\ ... \\ 0 \end{bmatrix}$ *or* $e_i = (0, 0, 0, ..., 0, 1, 0, ..., 0)^t$ and zero matrix $O = \begin{bmatrix} 0 \\ ... \\ 0 \end{bmatrix}$, one matrix

$$1 = \begin{bmatrix} 1 \\ ... \\ 1 \end{bmatrix} = \sum_{i=1}^{d} e_i$$

2.1.3 R^2 Vector space

The straight line on the coordinate plane is divided into subspace and affine subspace. A subspace is a straight line through which the sum of any two components of a straight line on the coordinate plane can enter the value in the straight line. A straight line passing through the origin is indicated by. The affine subspace is represented by a line that does not pass the origin in a straight line rather than a subspace. You can define dimensions with the affine subspace mentioned here. 0-dimensional affine subspace is point, 1-dimensional affine subspace is line, 2-dimensional affine subspace is plane, ... , The d-1 dimensional affine subspace is called the hyper plane.

2.1.4 Affine equation

$\lambda_1 x_1 + \lambda_2 x_2 + ... + \lambda_d x_d = 0$ is Affine equation. Just $\lambda_1 + \lambda_2 + ... + \lambda_d = 1$ $(0 \leq \lambda_1, \lambda_2, ..., \lambda_d \leq 1)$. That make $\lambda_d = 1 - (\lambda_1 + \lambda_2 + ... + \lambda_{d-1})$ so we can findλ_d by given $\lambda_1, ..., \lambda_{d-1}$. It is necessary to study about Convex hull.

2.1.5 Convex set

When a line segment connecting any two points belonging to a set of d-dimensional Euclidean spaces is included in the set, the set is called a convex set.
$[\vec{x}, \vec{y}] = \{\lambda\vec{x} + (1-\lambda)\vec{y} | 0 \leq \lambda \leq 1\}$ When the 1-dimension $[x, y]$ is $\lambda x + (1-x)y$, expand to 2-dimensional then $[(x, y), z]$ can expressed by $\lambda_1 x + \lambda_2 y + (1 - \lambda_1 - \lambda_2) = (\lambda_1 + \lambda_2)(\frac{\lambda_1}{\lambda_1+\lambda+2}x + \frac{\lambda_1}{\lambda_1+\lambda+2}) + (1 - \lambda_1 - \lambda_2)z//$

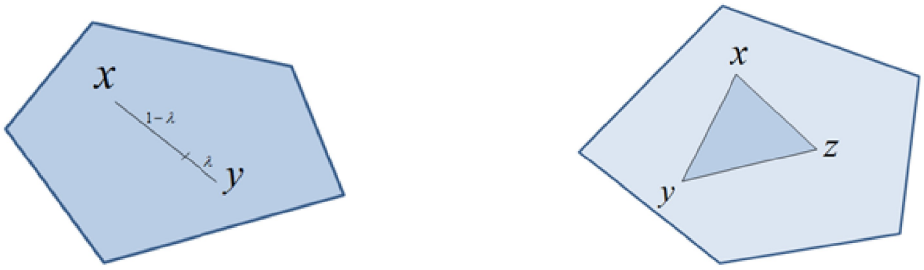

Figure 2: convex set in 1,2-dimension

2.1.6 Convex hull

Convex hull is the smallest figure that contains a given figure$K$ in any dimension. It satisfies the Convex set and is written as a symbol $conv(K)$. If the convex hull of k is $K^{'}$ then $K \subseteq R^d, K^{'} \subseteq R^d, K \subseteq K^{'}$. If the next one-dimensional figure like the left is K, the convex hull of this figure becomes the next right figure.

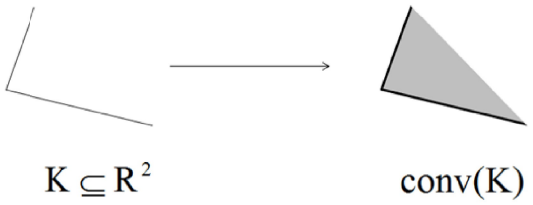

Figure 3: convex hull of 1-dimensional figure K

2.1.7 polytope

2.1.7.1 V-polytope

It is a polytope using a minimum dimension, that is, a polytope made using dots. The symbol is indicated as $conv(\{x_1, x_2, ..., x_k\})$.

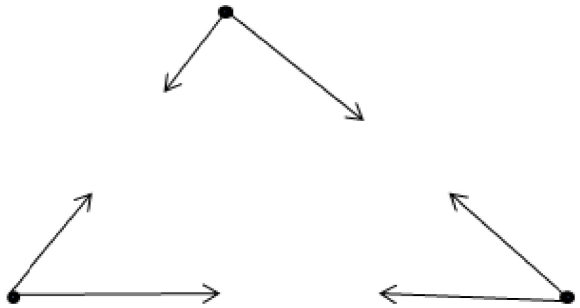

Figure 4: example of 2-dimensional V-polytope

2.1.7.2 H-polytope

It is a polytope that can be defined as the intersection of half spaces divided into hyperplanes. In contrast to V-polytope, which is defined as the smallest component, it is polytope, which is defined as the highest component.

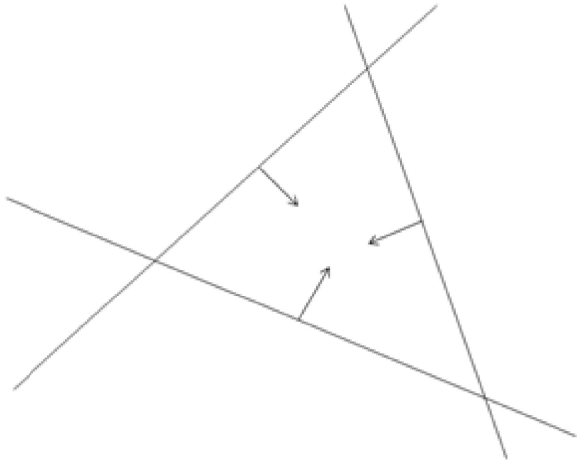

Figure 5: example of 2-dimensional H-polytope

As an example, the polytope of Figure 5 presented in the 2D plane can be defined as H-polytope. First, the line segment (d-1 dimensional component) of the proposed polytope is extended to make it a hyper plane. If you select the half spaces that contain the interior of the polytope from each half space divided by the created hyper plane, their intersection is H-polytope.

2.1.7.2 dimension of polytope

The dimension of the polygon P is the smallest dimension containing the polygon P.

2.2 Affinely Isomorphic and Simple figure

2.2.1 Definition of Affinely isomorphic

When $P \subseteq R^d, Q \subseteq R^e$, P, Q are Affinely isomorphic. if $f : R^d \to R^e$ in the Affinely map(the vector space with not defined zero point), their vertex are bijection.

2.2.2 Facet

(d-1)-dimensional polytope which make d-dimensional polygon.
ex) polygone → edge, polyhedron → face, polychoron → cell,..., n-polytope → (n-1)-polytope

2.2.3 Face

Intersection of two Hyperplane, sharing part of Facet.

2.2.4 Simplex

The figure which can make by at least Facet

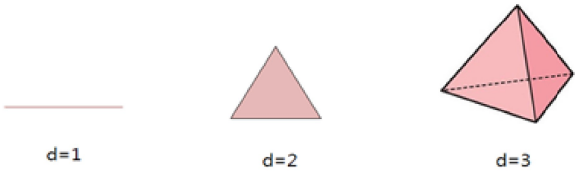

Figure 6: d=n dimensional simplex

(d+1)-dimension is needed to make the d-simplex.

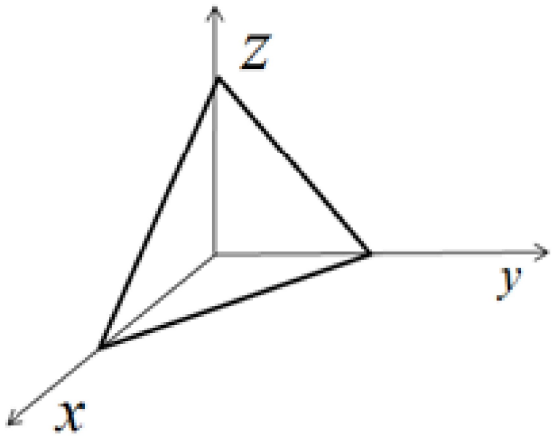

Figure 7: example of 2-dimensional simplex

$$\Delta_d := \{x \in R^{d+1} : ||x = 1, x_i \geq 0\} = conv\{e_1, ..., e_{d-1}\}$$

2.2.5 Cube

Figure which extended (d-1)dimensional polytope to d-dimensional axis.(extend to y-axis when $d = 1 \rightarrow d = 2$, extend to z-axis when $d = 2 \rightarrow d = 3$)

$C_d := \{x \in R^d : -1 \leq x_i \geq 1\} = conv\{(+1, -1)^d\}$

ex) $d = 1 \rightarrow \{+1, -1\}, d = 2 \rightarrow \{+1, -1\}^2, d = 3 \rightarrow \{+1, -1\}^3$

All vertex are included by the least d-facet and d-dimensional hypercube is called simple polytope.

Figure 8: d=n dimensional cube

2.2.6 Cross polytope

Cross polytope is the figure which have properties of cube and simplex. Dual polytope of Cube and facet of n-cross polytope is (n-1)-simplex.

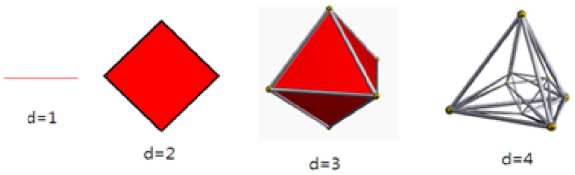

Figure 9: d=n dimensional cross polytope

$$C_d := \{x \in R^d : \sum |x_i| \geq 1\} = conv(\pm e_1, ..., e_d)$$

ex) $d = 1 \rightarrow \{+e, -e\}, d = 2 \rightarrow \{\pm e_1, \pm e_2\}, d = 3 \rightarrow \{\pm e_1, \pm e_2, \pm e_3\}$
d-cross polytope is composed by d-simplex's facet, so we called it simplicial(so each facet has at least vertex).

2.3 operation of Polytope

2.3.1 Pyramid

Start with established figure(polytope) ...(0)
making the vertex(point, 1-dimensional face) which is not exist in the axis of established figure(polytope) ...(1)
connect vertex which pointed at (1) and vertexes of established figure...(2)
make the convex hull with established figure by the connecting vertex...(3)

we called figure(polytope) which pyramid operated from the established figure to pyr(P).

2.3.2 Prism

Start with established figure(polytope) ...(0)
move the established figure to the axis which is not exist established...(1)
connect vertexes of figure which made at (1) and vertexes of established figure...(2)

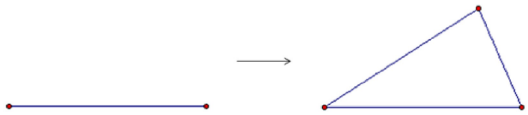

Figure 10: pyramid operation example 1

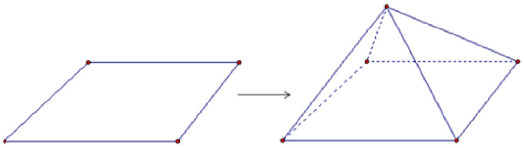

Figure 11: pyramid operation example 2

make the convex hull by the connecting vertex...(3)

we called figure(polytope) which prism operated from the established figure to prism(P). In the case of (1), degree of moving is not necessary. Because all of the case make affinely isomphic.

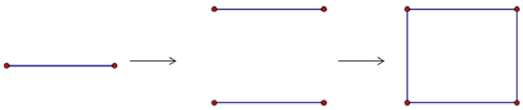

Figure 12: prism operation example 1

2.3.3 Minkowski sum(sum of polytope)

Choose one between two polytopes, $P, Q \subseteq R^d$ and $P + Q = \{x + y : x \in P, y \in Q\}$ (1)
Choose one vertex among the polytope which chose at (1). (2)
The vertices selected in (2) are matched by parallel translation to one vertex of the remaining polytope without taking in (1). (3)

The process of (3) is executed for all vertices of the remaining polytope without taking in (1). (4)
Take convex hull on the figure created through (4). (5)

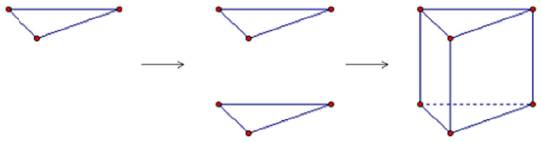

Figure 13: prism operation example 2

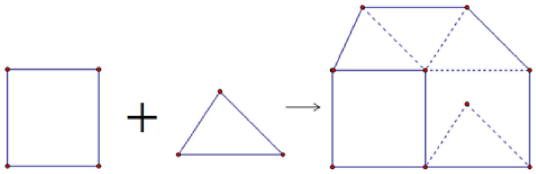

Figure 14: Minkowski sum example 2

2.4 Poset

2.4.1 Poset

Poset is an abbreviation of Partially Ordered Set, which is a set that cannot be compared in whole, but can be partially compared. Here, the comparison of size between sets can be expressed as an inclusion relationship between sets.

2.4.2 basic properties of Poset

For sets a, b and c
A) a⊆a
A set can be said to be greater than or equal to the set itself.
ex) 1,2,3 ⊆ 1,2,3
B) a⊆b and b⊆a ⇔ a = b
When a large or the same relationship is established between two sets, the two sets are considered as the same set.
ex) a⊆b and b⊆c ⇔ a⊆c
The relationship between two sets can be expressed by dividing the relationship between three sets.

2.4.3 analysis about Poset

Poset can be visually drawn using the Hasse diagram to figure out the distinct relationship between each component.

When the poset is represented by the Hasse diagram, the premise is that there is no set between all the segments connected by each subset.

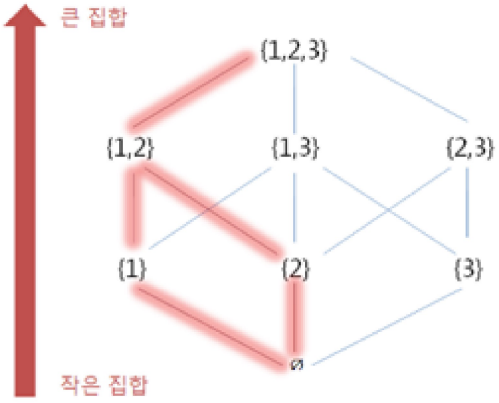

Figure 15: subset relation tree of the Poset1,2,3

if analyzing Poset1,2,3 with Hasse diagram
1. The larger the set, the upper part of the Hasse diagram and the smaller the lower part.
2. The largest set and the smallest set exist only one.
3. It is possible to compare large and small sets between the subsets 1,2 centered up and down (when Poset 1,2,3 is shown in the Hasse diagram)
4. Subsets 1,3, 2,3, 3 that do not correspond to 3 cannot be compared with 1,2.

2.4.4 Poset language

About a random set x, y
upper bound $=z(z^{'} \geq x^{'}, z \geq y)$ can be thought of as a concept similar to the common multiple, as two sets x, y, a larger set than simultaneously.
The least upper bound(join) $=w(w \geq x, w \geq y, z \geq w)$ is indicated by the symbol$x \bigvee y$, and it is the largest set of upper bounds and can be thought of as a concept similar to the least common multiple.
lower bound $=z^{'}(z^{'} \leq x^{'}, z^{'} \leq y^{'})$ can be thought of as a concept similar to the common divisor, with two sets and a smaller set at the same time.
The greatest lower bound (meet) $=w^{'}(w^{'} \leq x^{'}, w^{'} \leq y^{'}, w^{'} \leq z^{'})$is indicated by the symbol $x \bigwedge y$, and it is the smallest set of lower bounds and can be thought of as a concept similar to the greatest common divisor.

2.4.5 Lattice poset

Lattice poset is the Poset, which $x \bigwedge y, x \bigvee y$ are always exist for random set x, y
The polytope we deal with is a lattice poset where there is always one join and meet each, and all lattices in the face of the polytope are graded lattices with grades that

increase sequentially one dimension at a time.

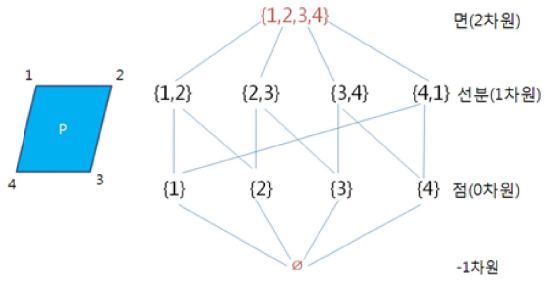

Figure 16: polytope P and p's poset

Analyzing the above polytope P as an example, the polytope P itself, that is, the $x \bigvee y$(join) of the two-dimensional face 1,2,3,4 polytope P, and the -1D $\emptyset$ is the $x \bigwedge y$(meet) of the polytope P. Here, $\emptyset$ is a concept created by necessity, which facilitates the operation of polytope. Since all polytopes are lattice poses, various polytope operations are possible and the results can be deduced. In addition, all faces of polytope P are faces 1,2,3,4, line segments 1,2, 2,3, 3,4, 4,1 and points 1, 2, 3, 4, $\emptyset$ show that it is a graded lattice due to the properties of polytope.

2.4.6 Dual polytope

Only when the faces of F_1 and F_2 of P satisfy that F_1 is part of F_2, while satisfying that faces of Q f (F_2) are part of f (F_1), between all faces of P and all faces of Q Two functions P and Q are said to be Dual when there is a function f to map one to one.

2.5 Introduction about Polymake

Polymake running on Ubuntu, a type of Linux operating system, is a program that can perform commands such as calculating convex set functions and visually displaying them in Geomview. This is a quick look at the commands used in Polymake.

Start Polymake: polymake; Create a new shape: $p = new Polytope ¡Rational¿;
Point setting: $p − > POINTS = ≪ ".";
Output facet count: $p− > N_FACETS;
simple or not: $p − > SIMPLE; (with or without simple polytope, face 1, face 0)
In addition, you can make shapes with characters such as cube and sphere.
$q = cube (3,0); − > (Specify 3D cube as q) $q − > VISUAL; -(The previous figure q is schematically expressed using geomview)
$points = new Matrix ¡Rational¿ ([[1, -1, -1], ..., [1,0,0]]);

It is possible to express matrices as $\begin{bmatrix} [1,-1,-1] \\ \vdots \\ [1,0,0] \end{bmatrix}$.
After that, p is defined as a new polytope with the above matrix as points with $p = new polytope ¡Rational¿ (POINTS => $points) ;.

Print $p-POINTS; => Display the matrix above as dots
Print $p-VERTICES; => Display the vertex of the convex hull of the above points
Print $q-FACETS; => (Shows FACET's equation) first term constant, others coefficient of axial component
Print $p− > F_VECTOR; => F_VECTOR output
Print $ p− > VOLUME => Area wanted (d = 2) Length wanted (d = 1)

3 content of research

3.1 Analogy of operation results

3.1.1 Pyramid

Because one additional point is taken, one point is added, and since the additionally taken point and the points of the existing polytope are connected to each other to form a line segment, the number of line segments that are one-dimensional components increases by the number of vertex elements of the existing polytope. In addition, since the face is newly generated from existing line segments, the number of faces that are two-dimensional components is increased by the number of one-dimensional components of the existing polytope. Since this process will be repeated, the result of pyramid operation can be inferred like this.

Components	Number	
	P(Conventional polytope)	pyr(P)
vertex	v	v+1
edge	e	e+v
2-dimensional face	f	f+e
⋮	⋮	⋮
k-dimensional face	n(k-dimensional face)	n(k-dimensional face) + n((k-1)-dimensional face))

Table 1: analogy of pyramid operation

The change in the number of vertex components is explained by taking a new vertex, which can also be explained by (0-1) - dimensional face, that is, the number of $\emptyset$ that mean the component of -1 dimension (one for every polytope).

3.1.2 Prism

Since each vertex is connected after moving, first, each component is doubled before calculation. After that, since the corresponding vertices are connected to each other, each line segment is generated, so the line segment increases by the number of existing vertices. Since this process will be repeated, the result of pyramid operation can be inferred like this.

Components	Number	
	P(Conventional polytope)	prism(P)
vertex	v	2v
edge	e	2e + v
2-dimensional face	f	2f+e
⋮	⋮	⋮
k-dimensional face	n(k-dimensional face)	2n(k-dimensional face) + n((k-1)-dimensional face))

Table 2: analogy of prism operation

3.1.3 Minkowski sum

1) Minkowski sum between 1-dimensional polytopes

Since one dimension can only be created up to two dimensions, it does not matter whether it is on the same plane or on another plane. If you look at two 1-dimensional segments as vectors with their respective directions, and perform Minkowski sum, you can get the result of a square.

However, when the directions of the vectors of the two straight lines are the same or opposite, the result of the Minlowski sum is not a two-dimensional square, but a one-dimensional segment.

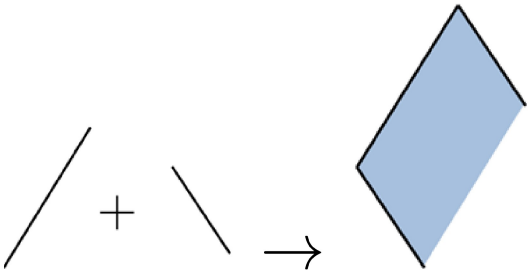

Figure 17: Result of Minkowski sum between 1-d

2) Minkowski sum between 1-dimensional polytope and 2-dimensional polytope

Minkowski sum results in one and two dimensions can occur up to three dimensions, so if you are on the same plane and on another plane, you should think of them in two ways.

a) On the same plane

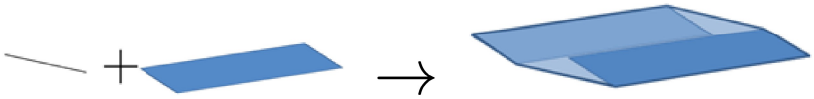

Figure 18: Result of Minkowski sum on the same plane

b) On the other plane

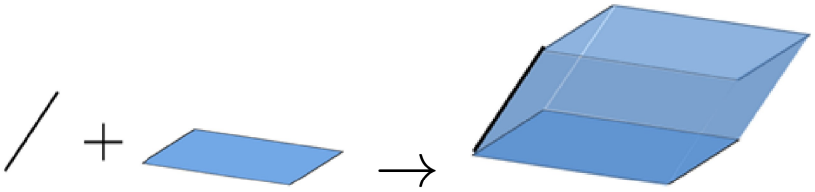

Figure 19: Result of Minkowski sum on the other plane

3) Minkowski sum between 2-dimensional polytope

When each line segment (one-dimensional component) that is the facet of a two-dimensional polytope is given a unit vector (length 1) in the normal direction toward the inside of the polytope, and the points of the vectors are collected at the origin, the convex hull of the end points is obtained. Combinatorically) the figure comes out. Here, when there are two or more vectors having the same direction, it is regarded as one.

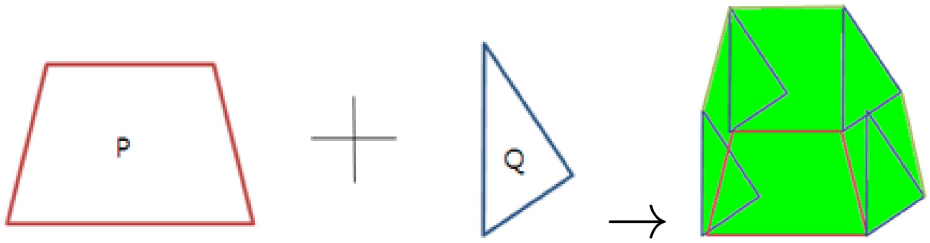

Figure 20: Result of Minkowski sum of P, Q

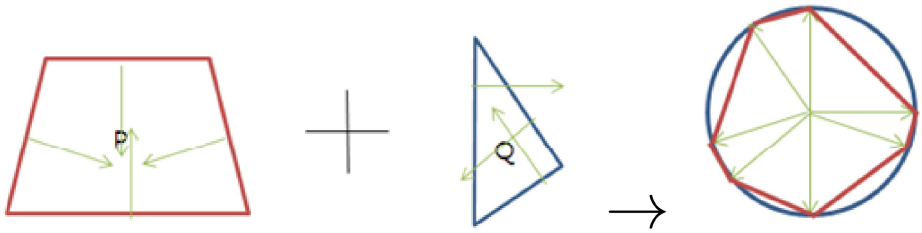

Figure 21: Result of Minkowski sum by analogy

The 2-dimensional polytope is affinely isomorphic if the number of vertices or lines is the same, so it is combinatorically the same as the actual Minkowski sum polytope.

3.2 Make the function of polytope operation in python

3.2.1 Pyramid

Basic Idea

1. define f_vector(n,v).
   1.1. n = Degree of dimension, v = Number of Vertex.
   1.2. if n >= v, return error.
   1.3. else if n == 2, define the f_vector of 2-dimension with array.
   1.4.1. else if n == 3 and v == 4, define the f_vector of 3-dimension with array.
   1.4.2 else if n == 3 and v>4,define the f_vector of 3-dimension with array.
   1.5 else define the f_vector of higher than 4-dimensional with Euler's function.

2. define pyramid(n, v):
   2.1. declare f_vector as a variable
   2.2. if n == 2, define the pyramid operation of 2-dimension with array and f_vector.
   2.3. else, define the pyramid operation of other dimension with array and f_vector.

3.2.2 Prism

Basic Idea

1. define f_vector(n,v).
   1.1. n = Degree of dimension, v = Number of Vertex.
   1.2. if n >= v, return error.
   1.3. else if n == 2, define the f_vector of 2-dimension with array.
   1.4.1. else if n == 3 and v == 4, define the f_vector of 3-dimension with array.
   1.4.2 else if n == 3 and v>4,define the f_vector of 3-dimension with array.
   1.5 else define the f_vector of higher than 4-dimensional with Euler's function.

2. define prism(n, v):
   2.1. declare f_vector as a variable
   2.2. if n == 2, define the prism operation of 2-dimension with array and f_vector.
   2.3. else, define the prism operation of other dimension with array and f_vector.

4 result of research

4.1 Result of operation with polymake

4.1.1 Pyramid

```
polytope > $a= 5;

polytope > $p = rand_sphere(2,$a);

polytope > print $p -> F_VECTOR;
polymake: used package lrs
  Implementation of the reverse search algorithm of Avis and Fukuda.
  Copyright by David Avis.
  http://cgm.cs.mcgill.ca/~avis/C/lrs.html

polymake: used package cdd
  cddlib
  Implementation of the double description method of Motzkin et al.
  Copyright by Komei Fukuda.
  http://www-oldurls.inf.ethz.ch/personal/fukudak/cdd_home/

5 5
polytope > print pyramid($p) -> F_VECTOR;
6 10 6
polytope > print pyramid(pyramid($p)) -> F_VECTOR;
polymake: used package ppl
  The Parma Polyhedra Library ([[wiki:external_software#PPL]]): A C++ library for convex polyhedra
  and other numerical abstractions.
  http://www.cs.unipr.it/ppl/

7 16 16 7
```

Figure 22: pyramid operation 1

After setting the value of $a to 5 and entering the command $p = rand_sphere(2, $a), a figure (pentagon) connecting 5 random points with a constant distance from a point in 2D space is generated. print If you enter the command $p− > F_VECTOR;, 5 and 5 appear from 0-dimensional components to 1-dimensional components, respectively. From the two-dimensional components, 6, 10, and 6 appear. If you print pyramid (pyramid ($ p))− > F_VECTOR again and calculate pyramid once again, 7, 16, 16, and 7 appear from 0 to 3 dimensional components respectively.

```
polytope > $a = 12;

polytope > $p = rand_sphere(8,$a);

polytope > print $p->F_VECTOR;
12 66 219 480 705 669 372 93
polytope > print pyramid($p)->F_VECTOR;
13 78 285 699 1185 1374 1041 465 94
polytope >
```

Figure 23: pyramid operation 2

After setting the value of $a to 12, if you enter the command $p = rand_sphere(8, $a), a figure is created that connects 12 random points with a constant distance from a point in 8-dimensional space. print If you input the command $p− > F_VECTOR;, 12, 66, 219, 480, 705, 669, 372, 93 from 0-dimensional components to 7-dimensional components appear respectively, and print pyramid ($p)− > F_VECTOR pyramid When F_VECTOR of the calculated figure $p is obtained, 13, 78, 285, 699, 1185, 1374, 1041, 465, and 94 from 0-dimensional components to 8-dimensional components, respectively. As a result, the result of the pyramid-calculated figure d-dimensional F_VECTOR is a general figure d-dimensional F_VECTOR + (d-1) -dimensional F_VECTOR.

4.1.2 Prism

```
polytope > $a= 18;

polytope > $p = rand_sphere(7,$a);

polytope > print $p->F_VECTOR;
18 150 682 1720 2382 1694 484
polytope > print prism($p)->F_VECTOR;
36 318 1514 4122 6484 5770 2662 486
```

Figure 24: pyramid operation 1

After setting the value of $a to 18, if you enter the command $p = rand_sphere(7, $a), a figure is created that connects 18 random points with a constant distance from a point in 7-dimensional space. print If you input the command $p− > F_VECTOR;, 18, 150, 682, 1720, 2382, 1694, 484 from 0-dimensional components to 6-dimensional components appear respectively, and print prism ($p)− > F_VECTOR prism When F_VECTOR of the calculated figure $p is obtained, 36, 318, 1514, 4122, 6484, 5770, 2662, and 486 from 0-dimensional components to 7-dimensional components, respectively. As a result, the result of the prism-calculated figure d-dimensional F_VECTOR is a general figure d-dimensional F_VECTOR * 2 + (d-1)-dimensional F_VECTOR.

4.1.3 Minkowski sum

1) Minkowski sum between 1-dimensional polytope

a) Minkowski sum of two non-parallel segments

```
polytope > $p=new Polytope<Rational>;

polytope > $p->POINTS=<<".";
polytope (2)> 1 0 0
polytope (3)> 1 1 0
polytope (4)> .

polytope > print $p->F_VECTOR;
polymake: used package cdd
  cddlib
  Implementation of the double description method of Motzkin et al.
  Copyright by Komei Fukuda.
  http://www-oldurls.inf.ethz.ch/personal/fukudak/cdd_home/

polymake: used package lrs
  Implementation of the reverse search algorithm of Avis and Fukuda.
  Copyright by David Avis.
  http://cgm.cs.mcgill.ca/~avis/C/lrs.html

2
polytope > $q=new Polytope<Rational>;

polytope > $q->POINTS=<<".";
polytope (2)> 1 0 0
polytope (3)> 1 1 1
polytope (4)> .

polytope > print $q->F_VECTOR;
2
polytope > $m = minkowski_sum($p,$q);

polytope > print $m->F_VECTOR;
4 4
polytope >
```

Figure 25: non-parallel segments' minkowski sum

If you create a segment $p that is {(0,0), (1,0)} in a two-dimensional space with the command $p − > POINTS = ≪ ".";, Look at the F_VECTOR of this segment, 0-dimensional-2 is In this way, the command $p − > POINTS = ≪ "." Creates a line segment $q {(0,0), (1,1)} in a two-dimensional space, and when looking at F_VECTOR, 0-dimensional-2 appears. . Command these two segments Minkowski sum with $m = minkowski_sum ($p, $q) and F_VECTOR is calculated to get 0D-4 1D-4. This shows that the Minkowski sum of two non-parallel segments is square.

b) Minkowski sum of parallel segments

```
polytope > $p=new Polytope<Rational>;

polytope > $p->POINTS=<<".";
polytope (2)> 1 0 0
polytope (3)> 1 1 0
polytope (4)> .

polytope > print $p->F_VECTOR
polytope (2)> ;
2
polytope > $p=new Polytope<Rational>;

polytope > $p->POINTS=<<".";
polytope (2)> 1 0 0
polytope (3)> 1 1 0
polytope (4)> .

polytope > print $p->F_VECTOR;
2
polytope > $q=new Polytope<Rational>;

polytope > $q->POINTS=<<".";
polytope (2)> 1 0 0
polytope (3)> 1 1 0
polytope (4)> .

polytope > print $q->F_VECTOR;
2
polytope > $m = minkowski_sum($p,$q);

polytope > print $m -> F_VECTOR;
2
```

Figure 26: parallel segments' minkowski sum

The segments $p and $q that are {(0,0), (1,0)} in a two-dimensional space with the commands $p − > POINTS = ≪ ".", $q − > POINTS = ≪ "." If you look at the F_VECTOR of these segments, 0D-2 appears. If these two segments are minkowski sum with command $m = minkowski_sum ($p, $q) and F_VECTOR is calculated, 0-dimensional-2 appears. This shows that the Minkowski sum of two parallel segments is a line segment.

2) Minkowski sum between 1-d and 2-d

a) On the same plane

```
polytope > $p = new Polytope<Rational>;

polytope > $p->POINTS=<<".";
polytope (2)> 1 0 0
polytope (3)> 1 1 1
polytope (4)> .

polytope > print $p->F_VECTOR;
polymake: used package cdd
  cddlib
  Implementation of the double description method of Motzkin et al.
  Copyright by Komei Fukuda.
  http://www-oldurls.inf.ethz.ch/personal/fukudak/cdd_home/

polymake: used package lrs
  Implementation of the reverse search algorithm of Avis and Fukuda.
  Copyright by David Avis.
  http://cgm.cs.mcgill.ca/~avis/C/lrs.html

2
polytope > $q = cube(2,0);

polytope > print $q->F_VECTOR;
4 4
polytope > $m=minkowski_sum($p,$q);

polytope > print $m->F_VECTOR;
6 6
```

Figure 27: on the same plane

If you create a segment $p that is $\{(0,0),(1,1)\}$ in a two-dimensional space with the command $p− >POINTS=<<”.”, if you look at F_VECTOR of this segment, 0-dimensional-2 is If you come out and enter $q as a two-dimensional cube-square, F_VECTOR of $q will appear 0-dimensional-4, 1-dimensional-4. If the above figures $p and $q are minkowski sum with the command $m = minkowski_sum($p,$q) and F_VECTOR is calculated, a hexagon of 0-dimensional-6 1-dimensional-6 appears. For this reason, it can be seen that a two-dimensional figure emerges when Minkowski sums a one-dimensional figure and a two-dimensional figure in the same plane.

b) On the other plane

```
polytope > $p = new Polytope<Rational>;

polytope > $p->POINTS=<<".";
polytope (2)> 1 0 0 0
polytope (3)> 1 0 1 0
polytope (4)> .

polytope > print $p->F_VECTOR;
2
polytope > $q=new Polytope<Rational>;

polytope > $q->POINTS=<<".";
polytope (2)> 1 0 0 0
polytope (3)> 1 1 0 0
polytope (4)> 1 1 1 1
polytope (5)> .

polytope > print $q->F_VECTOR
polytope (2)> ;
3 3
polytope > $m = minkowski_sum($p,$q);

polytope > print $m->F_VECTOR;
6 9 5
polytope >
```

Figure 28: on the other plane

Command polytope > $p− >POINTS=<<”.” Create a line segment $\{(0,0,0),(0,1,0)\}$ in 3D space with the command polytope> $q− >POINTS=<<”.” Enter $\{(0,0,0)(1,0,0)(1,1,1)\}$. When F_VECTOR is obtained, 0-dimensional-2, 0-dimensional-3, and 0-dimensional-3 are obtained, respectively. When F_VECTOR of their Minkowski sum is obtained, 0-dimensional-6 1-dimensional-9 2-dimensional-5 is obtained. Therefore, it can be seen that the Minkowski sum of 1-dimensional and 2-dimensional shapes that are not parallel to each other in the 3-dimensional space becomes a 3-dimensional shape.

3) Minkowski sum between 2-d

a) On the same plane

```
polytope > $p = cube(2);

polytope > print $p->F_VECTOR;
4 4
polytope > $q=simplex(2);

polytope > print $q->F_VECTOR;
polymake: used package lrs
  Implementation of the reverse search algorithm of Avis and Fukuda.
  Copyright by David Avis.
  http://cgm.cs.mcgill.ca/~avis/C/lrs.html

3 3
polytope > $m=minkowski_sum($p,$q);

polytope > print $m->F_VECTOR;
polymake: used package cdd
  cddlib
  Implementation of the double description method of Motzkin et al.
  Copyright by Komei Fukuda.
  http://www-oldurls.inf.ethz.ch/personal/fukudak/cdd_home/

5 5
```

Figure 29: on the same plane

The command $p = cube(2) creates a two-dimensional cube square and the command $q = simplex(2) creates a two-dimensional simplex equilateral triangle. When

these F_VECTORs are calculated, 0-dimensional-3 and 1-dimensional-3, 0-dimensional-4 and 1-dimensional-4 are obtained, respectively. It can be inferred that it is a pentagon. As a result, it can be seen that the Minkowski sum between two-dimensional figures of the same plane becomes a two-dimensional figure.

b) On the other plane

```
polytope > $a = 4
polytope (2)> ;

polytope > $p1 = rand_sphere(3,$a);

polytope > $p2 = rand_sphere(3,$a);

polytope > print $p1->F_VECTOR;
4 6 4
polytope > print $p2->F_VECTOR;
4 6 4
polytope > $m=minkowski_sum($p1,$p2);

polytope > print $m->F_VECTOR;
13 26 15
```

Figure 30: on the same plane

If the value of $a is set to 9 and then the command $p1 = rand_sphere(2,$a) is entered, a figure connecting 9 points with a certain distance from any one point in the 2D plane is generated. Similarly, after specifying $p2 and $p3, create a dual using the command polarize the convex hull of $p1 and $p2, and designate him as $q1. Similarly, $q2 and $q3 are specified. Since dual was designated as the new polytope, the polytope that had the same point before taking the dual had the same line segment after taking the dual. Therefore, inferring the result of taking the minkowski sum, the number of segments of the polytope minkowski sumping $p1 and $p2 is (9+6)+(9+7)-9. Then, using N_FACETS to find the actual result, you can see that the actual result is the same as the inferred result.

4.2 Result of making polytope operation function in python

4.2.1 basic code

1) basic code

We created the following code according to the basic idea.

a) Make F_vector in low dimension

```
def f_vector(n, v):
    if n >= v:
        return "ERROR: rand_sphere: 2 <= dim < #vertices"
    else:
        ar=[]
        e = 0
        f = 0
        if n==2:
            ar.append(v)
            e += v
            ar.append(e)
            return ar

        elif n==3:
            ar.append(v)
            if v==4:
                e += 2*v-2
                f += v
                ar.append(e)
                ar.append(f)
                return ar
            elif v > 4:
                e += 3*v-6
                f += 2*v-4
                ar.append(e)
                ar.append(f)
                return ar

        elif n==4:
            mx = []
            ver=0
            x=0
            ar.append(v)
            for _ in range(0,n):
                mx.append([])

            for i in range(1,v+1):
                ver+=i
                mx[0].append(i)

            for j in range(1,v+1):
                if j<=1:
                    mx[1].append(0)
                else:
                    e+=mx[0][j-2]
                    mx[1].append(e)
            ar.append(e)

            for k in range(1,v+1):
                if k<=2:
                    mx[2].append(0)
                else:
                    f+=mx[1][k-2]
                    mx[2].append(f)
            ar.append(f)

            for l in range(1,v+1):
                if l<=3:
                    mx[3].append(0)
                else:
                    x+=mx[2][l-2]
                    mx[3].append(x)
            ar.append(x)

            return ar
```

Figure 31: F_vector in low dimension

b) Make F_vector in high dimension

```
def f_vector2(n, v):
    if n >= v:
        return "ERROR: rand_sphere: 2 <= dim < #vertices"
    else:
        ar=[]
        e = 0
        f = 0
        if n==2:
            ar.append(v)
            e += v
            ar.append(e)
            return ar
        elif n==3:
            ar.append(v)
            if v==4:
                e += 2*v-2
                f += v
                ar.append(e)
                ar.append(f)
                return ar
            elif v > 4:
                e += 3*v-6
                f += 2*v-4
                ar.append(e)
                ar.append(f)
                return ar
        else :
            sum_e = 0
            sum_f = 0
            last = 0
            # vetex
            f_v = f_vector2(n-1, n)[0] + v-n
            ar.append(f_v)
            # edege
            f_v = f_vector2(n-1, n)[0] + v-n
            ar.append(f_v)
            # edege
            for i in range(n,v):
                sum_e+=i
            f_e = f_vector2(n-1, n)[1] + sum_e
            ar.append(f_e)
            # 면부터 다포체
            for i in range(2,n-1):
                if (v-n == 1):
                    x=0
                    x=f_vector2(r-1,n)[i-1]+f_vector2(n-1,n)[i]
                    ar.insert(i,x)
                elif (v-n >= 2):
                    y=0
                    y=f_vector2(r,v-1)[i-1]+f_vector2(n,v-1)[i]
                    ar.insert(i,y)
            # 마지막 수, 오일러 함수 이용
            if n%2 == 0:
                for i in range(0,n-1):
                    last += ar[i]*(-1)**i
                ar.append(last)
            else:
                for i in range(0,n-1):
                    last += ar[i]*(-1)**i
                last = 2 - last
                ar.append(last)
            return ar
```

Figure 32: F_vector in high dimension

c) Pyramid Operation in python

```
def pyramid(n, v):
    rs = []
    ar = f_vector(n, v)
    i=0
    if n==2:
        rs.append(ar[0]+1)
        rs.append(ar[1]+ar[0])
        rs.append(1+ar[1])
    else:
        rs.append(ar[0]+1)
        for i in range(1, n):
            rs.append(ar[i]+ar[i-1])
        rs.append(1+ar[n-1])
    return rs
```

Figure 33: pyramid in python

d) Prism Operation in python

```
def prism(n, v):
    rs = []
    ar = f_vector(n, v)
    i=0
    if n==2:
        rs.append(2*ar[0])
        rs.append(2*ar[1]+ar[0])
        rs.append(2+ar[1])
    else:
        rs.append(2*ar[0])
        for i in range(1, n):
            rs.append(2*ar[i]+ar[i-1])
        rs.append(2+ar[n-1])
    return rs
```

Figure 34: prism in python

4.2.2 Checking with Analogy

a) Polytope operation in low dimension

```
f_vector(3, 12)
```

```
[12, 30, 20]
```

```
pyramid(3, 12)
```

```
[13, 42, 50, 21]
```

```
prism(3, 12)
```

```
[24, 72, 70, 22]
```

Figure 35: result of operation in python

```
polytope > $a = 12;

polytope > $p = rand_sphere(3, $a);

polytope > print $p -> F_VECTOR;
12 30 20
polytope > print pyramid($p) -> F_VECTOR;
13 42 50 21
polytope > print prism($p) -> F_VECTOR;
24 72 70 22
```

Figure 36: same result in Polymake

b) Polytope operation in high dimension

```
f_vector2(8, 9)
[9, 36, 84, 126, 126, 84, 36, 9]

pyramid(8, 9)
[10, 45, 120, 210, 252, 210, 120, 45, 10]

prism(8, 9)
[18, 81, 204, 336, 378, 294, 156, 54, 11]
```

Figure 37: result of operation in python

```
polytope > $a = 9;

polytope > $p = rand_sphere(8, $a);

polytope > print $p -> F_VECTOR;
9 36 84 126 126 84 36 9
polytope > print pyramid($p) -> F_VECTOR;
10 45 120 210 252 210 120 45 10
polytope > print prism($p) -> F_VECTOR;
18 81 204 336 378 294 156 54 11
```

Figure 38: same result in Polymake

5 conclusion

5.1 conclusion

a. The result of Pyramid operation of any Polytope can be inferred.
b. The result of Prism calculation of any Polytope can be inferred.
c. The results of Mikowski sum of arbitrary one-dimensional polytope and one-dimensional polytope can be inferred.
d. The results of Mikowski sum of arbitrary two-dimensional polytope and one-dimensional polytope can be inferred.
e. The results of Mikowski sums between two-dimensional polytopes can be inferred on the same plane.
f. The operation result confirmed by Polymake is the same as the inferred result.
g. F_vector generation and Pyramid and Prism operations were implemented through Python.
h. Verify the result of Polytope operation in python is same as the inferred result.

5.2 Suggestion

a. The results of the Minkowski sum could not be extended to n dimensions.
b. However, when we look at the minkowski sum results of two-dimensional polytopes on different planes, we can see that the face vector is maintained.

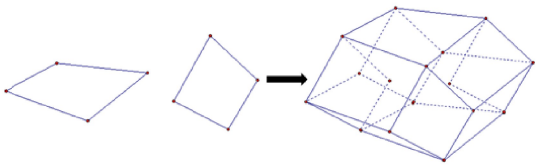

Figure 39: Minkowski sum in other plane

In this way, it seems that we can infer the results using the vector being maintained.

c. When creating f_vector in python, only the largest value, f_vector, was created, so it was impossible to check the value in polymake.

d. We tried to implement Minkowski sum in python, but it failed because there was no data for high-level shape calculation.

reference

◇ Guinter M. Ziegler, Lectures on Polytope, Springer-Verlag, 1997
◇ Branko Grunbaum, Convex polytopes, KSPRINGER, 2003