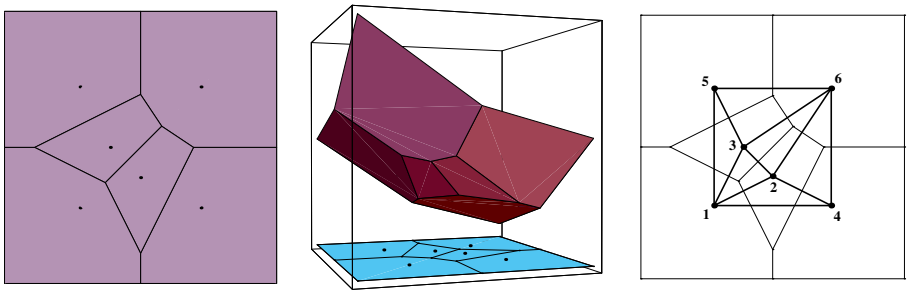
Frequently Asked Questions in Polyhedral Computation

Komei Fukuda
Department of Mathematics
Swiss Federal Institute of Technology, Lausanne, and
Institute for Operations Research
Swiss Federal Institute of Technology, Zürich, Switzerland
fukuda@ifor.math.ethz.ch

August 26, 1998



Contents

1 What is Polyhedral Computation FAQ?

This is an FAQ to answer some basic questions arising from certain geometric computation in general dimensional (mostly Euclidean) space. The main areas to be covered are the convex hull computation of a finite point set, the vertex enumeration for a convex polytope, the computation of Voronoi diagram and Delaunay triangulation, in R^d. We illustrate typical solution processes with small examples and publicly available codes such as cdd+ and lrs.

It is still very incomplete and perhaps contains a number of typos and mistakes at this moment, but I will try to make it as complete as possible for the primary purposes.

2 Convex Polyhedron

2.1 What is a convex polytope/polyhedron?

A subset P of R^d is called a *convex polyhedron* if it is the set of solutions to a finite system of linear inequalities, and called *convex polytope* if it is a convex polyhedron and bounded. When a convex polyhedron (or polytope) has dimension k, it is called a *k-polyhedron* (*k-polytope*). For the sequel, we might omit **convex** for convex polytopes and polyhedra, and call them simply polytopes and polyhedra.

2.2 What is a face of a convex polytope/polyhedron?

Let P be a convex d-polyhedron (or d-polytope) in R^d.

For a real d-vector c and a real number d, a linear inequality $c^T x \leq d$ is called *valid* for P if $c^T x \leq d$ holds for all $x \in P$. A subset F of a polyhedron P is called a *face* of P if it is represented as

$$F = P \cap \{x : c^T x = d\}$$

for some valid inequality $c^T x \leq d$.

To define faces geometrically, it is convenient use supporting hyperplanes. A hyperplane h of R^d is supporting P if one of the closed halfspaces of h contains P. A subset F of P is called a *face* of P if it is either $\emptyset$, P itself or the intersection of P with a supporting hyperplane.

The faces of dimension 0, 1, $d-2$ and $d-1$ are called the *vertices*, *edges*, *ridges* and *facets*, respectively. The vertices coincide with the *extreme points* P which are defined as points which cannot be represented as convex combinations of two other points in P. When an edge is not bounded, there are two cases: either it is a line or a half-line starting from a vertex. A half-line edge is called an *extreme ray*.

2.3 What is the face lattice of a convex polytope

The *face poset* $FL(P)$ of a convex polyhedron is the set of all faces of P ordered by set inclusion. Two polytopes are called *isomorphic* if their face posets are isomorphic. The face poset of a convex polytope is a lattice.

2.4 What is a simplex?

A subset P of R^d is called a *k-simplex* ($k = 0, 1, 2, \ldots$) if it is the convex hull of $k+1$ affinely independent points. It has exactly $k+1$ vertices and $k+1$ facets. A simplex is a k-simplex for some k.

2.5 What is a hypercube?

A subset P of R^d is called a *unit d-cube* if it is the convex hull of all 2^d points with components 0 or 1. It has exactly 2^d vertices and $2d$ facets. A *cube* is a convex polytope which is isomorphic to the k-cube for some k.

2.6 What is convex hull? What is convex hull computation?

For a subset S of R^d, the convex hull $conv(S)$ is defined as the smallest convex set in R^d containing S.

The convex hull computation means the "determination" of $conv(S)$ for a given finite set of n points $S = \{p^1, p^2, \ldots, p^n\}$ in R^d.

The usual way to determine $conv(S)$ is to represent it as the intersection of halfspaces, or more precisely, as a set of solutions to a minimal system of linear inequalities. This amounts to output a matrix $A \in R^{m \times d}$ and a vector $b \in R^m$ for some m such that $conv(S) = \{x | A\ x \leq b\}$.

Some people called the convex hull computation as the determination of extreme points of $conv(S)$, or equivalently that of redundant points in S to determine $conv(S)$. This is much simpler computation than the usual definition. In fact, this can be done by solving n linear programs 4.

2.7 What is the Minkowski-Weyl theorem for convex polyhedra?

The Minkowski-Weyl Theorem states every polyhedron or polytope is finitely generated. More precisely, for two subsets P and Q of R^d, $P+Q$ denotes the *Minkowski sum* of P and Q:

$$P+Q \;=\; \{p+q : p \in P \text{ and } q \in Q\}.$$

Theorem 1 (Minkowski-Weyl's Theorem) *For a subset P of R^d, the following statements are equivalent:*

(a) *P is a polyhedron, i.e., for some real (finite) matrix A and real vector b, $P = \{x : Ax \leq b\}$;*

(b) *There are finite real vectors $v_1, v_2, \ldots, v_n$ and $r_1, r_2, \ldots, r_s$ in R^d such that $P = conv(v_1, v_2, \ldots, v_n) + nonneg(r_1, r_2, \ldots, r_s)$.*

Thus, every polyhedron has two representations of type (a) and (b), known as (halfspace) *H-representation* and (vertex) *V-representation*, respectively.

2.8 What is the vertex enumeration problem, and what is the facet enumeration problem?

When a polyhedron P in R^d has at least one extreme point and full dimensional, both representations (a) and (b) in Miknowski-Weyl Theorem 1 are unique up positive multiples of each inequality and ray r_j.

Under these regularity conditions, the conversions between the H-representation and the V-representation are well-defined fundamental problems. The transformation (a) to (b) is known as the *vertex enumeration* and the other (b) to (a) is known as the *facet enumeration*. When P is in addition bounded (i.e. polytope), the facet enumeration problem reduces to what we call the convex hull problem, see 2.6.

If a given polyhedron does not satisfy the assumptions, it is easy to transform the polyhedron to an isomorphic lower dimensional polyhedron satisfying the assumptions.

There are easy (nondegenerate) cases and difficult (degenerate) cases. For simplicity, we assume P is bounded (i.e. polytope). The vertex enumeration is called *nondegenerate* if there is no point $x \in R^d$ which satisfies $d+1$ given inequalities with equality, and *degenerate* otherwise. The facet enumeration is called *nondegenerate* if there is no $(d+1)$ given points which are on a common hyperplane, and *degenerate* otherwise.

2.9 What is a cell complex? What is a triangulation?

A *cell complex* in R^d is a set K of convex polyhedra (called *cells*) in R^d satisfying two conditions: (1) Every face of a cell is a cell, and (2) If P and P' are cells, then their intersection is a common face of both.

3 Voronoi Diagram and Delaunay Triangulation

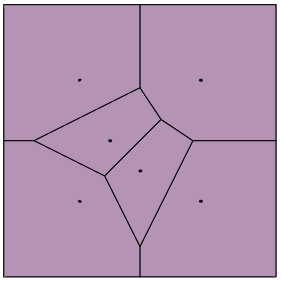
3.1 What is Voronoi diagram in R^d?

See also 3.2.

Given a set S of n distinct points in R^d, Voronoi diagram is the partition of R^d into n polyhedral regions $vo(p)$ ($p \in S$). Each region $vo(p)$, called the *Voronoi cell* of p, is defined as the set of points in R^d which are closer to p than to any other points in S, or more precisely,

$$vo(p) = \{x \in R^d | dist(x,p) \leq dist(x,q) \quad \forall q \in S - p\},$$

where $dist$ is the Euclidean distance function. (One can use different distance functions to define various variations of Voronoi diagrams, but we do not discuss them here.)

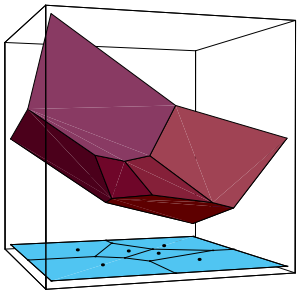


The set of all Voronoi cells and their faces forms a cell complex. The vertices of this complex are called the *Voronoi vertices*, and the extreme rays (i.e. unbounded edges) are the *Voronoi rays*. For each point $v \in R^d$, the *nearest neighbor set* $nb(S,v)$ of v in S is the set of points $p \in S - v$ which are closest to v in Euclidean distance. Alternatively, one can define a point $v \in R^d$ to be a *Voronoi vertex* of S if $nb(S,v)$ is maximal over all nearest sets.

In order to compute the Voronoi diagram, the following construction is very important. For each point p in S, consider the hyperplane tangent to the paraboloid in R^{d+1}: $x_{d+1} = x_1^2 + \cdots + x_d^2$. This hyperplane is represented by $h(p)$:

$$\sum_{j=1}^{d} p_j^2 - \sum_{j=1}^{d} 2p_j x_j + x_{d+1} = 0.$$

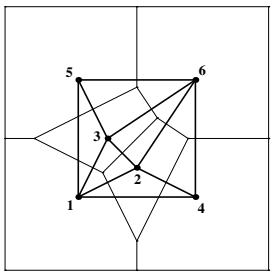
By replacing the equality with inequality $\geq$ above for each point p, we obtain the system of n inequalities, which we denote by $b - Ax \geq 0$. The polyhedron P in R^{d+1} of all solutions x to the system of inequalities is a lifting of the Voronoi diagram to one higher dimensional space. In other words, by projecting the polyhedron P onto the original R^d space, we obtain the Voronoi diagram in the sense that the projection of each facet of P associated with $p \in S$ is exactly the voronoi cell $vo(p)$. The vertices and the extreme rays of P project exactly to the Voronoi vertices and the rays, respectively.



3.2 What is the Delaunay triangulation in R^d?

See also 3.1, 2.9.

Let S be a set of n points in R^d. The convex hull $conv(nb(S, v))$ of the nearest neighbor set of a Voronoi vertex v is called the Delaunay cell of v. The Delaunay complex (or triangulation) of S is a partition of the convex hull $conv(S)$ into the Delaunay cells of Voronoi vertices.



The Delaunay complex is not in general a triangulation but becomes a triangulation when the input points are *nondegenerate*, i.e. no $d+2$ points are cospherical or equivalently there is no point $c \in R^d$ whose nearest neighbor set has more than $d+1$ elements.

The Delaunay complex is dual to the Voronoi diagram 3.1 in the sense that there is a natural bijection between the two complexes which reverses the face inclusions.

There is a direct way to represent the Delaunay complex, just like the Voronoi diagram 3.1. In fact, it uses the same paraboloid in R^{d+1}: $x_{d+1} = x_1^2 + \cdots + x_d^2$. Let $f(x) = x_1^2 + \cdots + x_d^2$, and let $\tilde{p} = (p, f(x)) \in R^{d+1}$ for $p \in S$. Then the so-called lower hull of the lifted points $\tilde{S} := \{\tilde{p} : p \in S\}$ reprents the Delaunay complex. More precisely, let

$$P = conv(\tilde{S}) + nonneg(e^{d+1})$$

where e^{d+1} is the unit vetor in R^{d+1} whose last component is 1. Thus P is the unbounded convex polyhedron consisting of $conv(\tilde{S})$ and any nonnegative shifts by the "upper" direction r. The nontrivial claim is that the the boundary complex of P projects to the Delaunay complex: any facet of P which is not parallel to the vertical direction r is a Delaunay cell once its last coordinate is ignored, and any Delaunay cell is represented this way.

3.3 Computing the Delaunay complex and the Voronoi diagram. What does it mean and how to do it with available software?

Let S be a given set of n points in R^d. Computing the Voronoi diagram normally means to generate the set $Vo(S)$ of Voronoi vertices, and computing the Delaunay complex is essentially the same thing. Once the Voronoi vertices are generated, the nearest neighbor sets $nb(S, v)$ for all Voronoi vertices v can be easily computed, and in fact most of the algorithms for generating the Voronoi vertices computes the nearest neighbor sets as well at the same time.

The complexity of computing the Voronoi diagram is not well understood in general. For the much easier nondegenerate case, there is an algorithm, known as the reverse search algorithm, which runs in time $O(nd|Vo(S)|)$.

How large is the number —Vo(S)— of output? The tight upper bound was given in [Sei91] which is $O(n^{\lfloor (d+1)/2 \rfloor})$. While this bound may be a far over-estimate of expected behavior, the number of output typically grows exponentially in n and d, and thus the computation itself is expected to be heavy. Therefore, one must take a caution to do the Delaunay/Voronoi computation. In fact,

> I know quite a few people who tried to use Voronoi diagram computation codes in order to accomplish a much simpler task.

It is not only a waste of time and computer resources, but it often leads to a prohibitively hard computation, while an appropriate use of mathematical techniques resolves the problem instantly.

For example, the following computations are much simpler and should be solved via linear programming techniques 4:

- For given two points p and q in S, check whether their Voronoi cells are adjacent in the Voronoi diagram, see 3.4.
- For any given point $c \in R^d$, find a Delaunay cell containing c, see 3.6.

The most natural way to compute the Voronoi diagram is by computing the vertices and the extreme rays of the polyhedron in R^{d+1} given in 3.1. By ignoring the last component of each vertices we obtain the Voronoi vertices.

3.3.1 Sample session with cdd+

Consider a simple two dimensional case: $d = 2$, $n = 6$ and $S = \{(0, 0), (2, 1), (1, 2), (4, 0), (0, 4), (4, 4)\}$. In principle the session below will work in any d and n, although the computation time depends heavily on the size.

The first step is to write down the system of linear inequalities in $(d + 1)$ variables as explained in 3.1: for each $p \in S$,

$$\sum_{j=1}^{d} p_j^2 - \sum_{j=1}^{d} 2p_j x_j + x_{d+1} \geq 0.$$

For our example, we have:

$$\begin{aligned}
0 \qquad\qquad\quad &+ x_3 \geq 0 \\
5 - 4x_1 - 2x_2 &+ x_3 \geq 0 \\
5 - 2x_1 - 4x_2 &+ x_3 \geq 0 \\
16 - 8x_1 \qquad &+ x_3 \geq 0 \\
16 \qquad -8x_2 &+ x_3 \geq 0 \\
32 - 8x_1 - 8x_2 &+ x_3 \geq 0
\end{aligned}$$

We denote by P the polyhedron of all solutions $x \in R^d$ satisfying the inequalities above. Now we prepare an input file of cdd+. The file must be in *polyhedra* format and for the system above, it is rather straightforward since it essentially codes the coefficients of the system.

```
* filename: vtest_vo.ine
H-representation
begin
 6   4   integer
  0  0  0  1
  5 -4 -2  1
  5 -2 -4  1
 16 -8  0  1
 16  0 -8  1
 32 -8 -8  1
end
incidence
input_adjacency
```

The last two lines "incidence" and "input_adjacency" are options for cdd+. They are not necessary for listing the vertices of the polyhedron, but but can be used to generate the nearest neighbor sets and the adjacency of Voronoi cells.

Now, by running cdd+ with commands:

```
% cddr+ vtest.ine
```

or

```
% cddf+ vtest.ine
```

we obtain three files: vtest_vo.ext, vtest_vo.iad and vtest_vo.ecd. Note that cddr+ runs in rational (exact) arithmetic and cddf+ runs in floating-point arithmetic. cddf+ runs much faster than cddr+ but it may not give a correct answer.

The file vtest_vo.ext would be something like the following:

```
*FINAL RESULT:
*Number of Vertices =6, Rays =4
begin
10  4  rational
```

```
 0 -1 0 0
 1 -3/2 2 0
 1 5/6 5/6 0
 1 2 -3/2 0
 0 0 -1 0
 1 27/10 27/10 56/5
 1 15/4 2 14
 0 1 0 8
 0 0 1 8
 1 2 15/4 14
end
hull
```

The output contains all the vertices and extreme rays of the (unbounded) polyhedron P in R^3. Namely each row starting with "1" represents a vertex. So the second row

```
  1 -3/2 2 0
```

represents the vertex $(-3/2, 2, 0)$. Each row starting with "0" represents an extreme ray, e.g. the first row

```
  0 -1 0 0
```

represents the ray $(-1, 0, 0)$.

By ignoring the last components, we obtain the set of six Voronoi vertices $(-3/2, 2)$, $(5/6, 5/6)$, $(2, -3/2)$, $(27/10, 27/10)$, $(15/4, 2)$ and $(2, 15/4)$ and four Voronoi rays $(-1, 0)$, $(0, -1)$, $(1, 0)$ and $(0, 1)$.

With the option "incidence, cdd+ outputs vtest_vo.ecd file:

```
begin
  10  6  7
 3 :  1 5 7
 3 :  1 3 5
 3 :  1 2 3
 3 :  1 2 4
 3 :  1 4 7
 3 :  2 3 6
 3 :  2 4 6
 3 :  4 6 7
 3 :  5 6 7
 3 :  3 5 6
end
```

Each row corresponds to the same row in vtest_vo.ine file. For example, the second data

```
  3 :  1 3 5
```

says the second data in vtest_vo.ine file:

```
1 -3/2 2 0
```

is a voronoi vertex whose nearest neighbor set is $\{p^1, p^3, p^5\}$. Also, this set corresponds to a Delaunay cell. Similarly, the first row

```
3 :  1 5 7
```

indicates the ray (the first output in vtest_vo.ine file)

```
0 -1 0 0
```

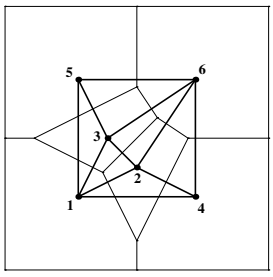
is determined by 1, 5 and 7th halfspaces. The 7th halfspace is an artificial one corresponding to the infinity. So this ray is determined by the input points 1 and 5 and going to infinity.

Thus, the index sets (triples, in this case) not containing the infinity 7 determine all Delaunay cells, and those containing 7 correspond to the Voronoi rays.

Finally, look at the vtest_vo.iad file:

```
begin
  7
 1 5 : 2 3 4 5 7
 2 4 : 1 3 4 6
 3 4 : 1 2 5 6
 4 4 : 1 2 6 7
 5 4 : 1 3 6 7
 6 5 : 2 3 4 5 7
 7 4 : 1 4 5 6
end
```

This file contains the graph structure of the Delaunay complex and equivalently the adjacency of Voronoi cells in the Voronoi diagram.



The first line in this file:

```
1 5 : 2 3 4 5 7
```

says the point p^1 is adjacent to 5 neighbors p^2, p^3, p^4, p^5 and p^7. Here, the point p^7 is the artificial infinity point which is considered adjacent to any input point whose Voronoi cell is unbounded.

As we remarked before, this graph information can be computed much more efficiently by linear programming. See 3.4.

3.4 Is it possible to compute only the adjacencies of Voronoi cells in the Voronoi diagram efficiently?

Yes, it can be done very efficiently by linear programming (LP), and very importantly this can be done for very large scale problems, with practically no bounds on the size with an efficient LP solver.

The method is simple. The lifting technique we described in 3.1 immediately gives the idea. Recall that the Voronoi diagram of a set S of n points in R^d is the projection of the following $(d+1)$-polyhedron to R^d space of the first d components.

$$P = \{x \in R^{d+1} \mid \sum_{j=1}^{d} p_j^2 - \sum_{j=1}^{d} 2p_j x_j + x_{d+1} \geq 0 \quad \forall p \in S\}.$$

For simplicity, denote it as

$$P = \{x \in R^{d+1} \mid b - A\, x \geq 0\},$$

where A is a given $n \times (d+1)$ matrix and b is a n-vector. Now for each $i = 1, \ldots, n$, consider the ith facet F_i of P:

$$F_i = \{x \in R^{d+1} \mid b - A\, x \geq 0 \text{ and } b_i - A_i\, x \leq 0\}, \tag{1}$$

Two facets F_i and F_j are called *adjacent* if the intersection $F_i \cap F_j$ is a facet of both, i.e. has dimension $d-2$. An equivalent definition is: they are *adjacent* if (*) the facet F_i becomes larger once the facet F_j is removed from the polyhedron, i.e. the jth inequality is removed from the system $b - A\, x \geq 0$.

It is easy to see that two Voronoi cells $vo(p^i)$ and $vo(p^j)$ are adjacent if and only if the corresponding facets F_i and F_j are adjacent in the polyhedron P. Now, we formulate the following LP for any distinct $i, j = 1, 2, \ldots, n$:

$$\begin{array}{lrcll} \text{minimize} & f(x) := & b_j & - & A_j \quad x \\ \text{subject to} & & b' & - & A \quad x \geq 0 \\ & & b_i & - & A_i \quad x \leq 0, \end{array} \tag{2}$$

where b' is equal to b except for jth component $b'_j = b_j + 1$. The new inequality system $b' - A\ x \geq 0$ is simply a modification of the original system obtained by relaxing the jth inequality a little bit. An important remark is, by definition (*), F_j and F_i are adjacent if and only if the objective value $f(x)$ is negative at an optimum solution. Thus we formulated the Voronoi adjacency computation as an LP problem.

How much do we gain by using LP for the adjacency computation, instead of computing the whole Voronoi diagram? A lot. It is hard to exaggerate this, because the LP (2) (in fact any LP) is solvable in polynomial time, whereas the associated Voronoi computation is exponential in d and n. Using the standard simplex method, the time complexity of solving an LP is not polynomial, but the practical complexity is roughly $O(nd^3)$.

3.4.1 Sample session with cdd+

With cdd+, a setup for computing the adjacency of Voronoi cells is quite simple. Consider the same example 3.3.1. For each input point $i = 1, 2, 3, 4, 5, 6$, we write the inequality system for the facet F_i:

$$\begin{array}{ccccc} b & - & A & x \geq 0 \text{ and} \\ b_i & - & A_i & x \leq 0, \end{array}$$

instead of writing the relaxed inequality (2). For example, for $i = 4$, we have

---

```
H-representation
begin
 7   4   real
  0  0  0  1
  5 -4 -2  1
  5 -2 -4  1
 16 -8  0  1
 16  0 -8  1
 32 -8 -8  1
-16  8  0 -1  % negative of 4th inequality
end
facet_listing
```

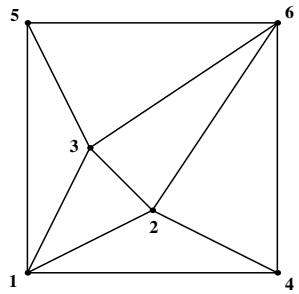
The last inequality is the negative of the forth inequality to force the forth inequality to be equality.

The code cdd+ has an option called "facet_listing". If we apply this option to the facet F_4, then cdd+ will check which of the given inequalities is redundant or not (essential), by soving the associated LP's (2) for each inequality j. As we discussed in 3.4, each inequality for F_4, except for the 4th and the 7th one,

The program cdd+ will output a file:

```
*Facet listing is chosen.
* `e` means essential and `r` means redundant.
begin
1 e: 2 7 1
2 e: 2 7 1
3 r: 2 7 6
4 e: 4 2 6
5 r: 7 2 6
6 e: 7 2 6
7 e: 7 2 6
end
```

We simply ignore the 4th and the 7th row, and also the lists of numbers after colons. Then we can consider the set $\{1, 2, 6\}$ of essential constraints as the set of indices of Voronoi cells adjacent to the 4th cell. Of course, this adjacency coincides with the adjacency of input points in the Delaunay triangulation. See the figure below.



3.5 Is it possible to compute only the edges of the Delaunay complex (triangulation) ?

This is essentially the same question as computing the adjacencies of Voronoi cells, see 3.4.

3.6 Is it possible to determine the Delaunay cell containing a given point efficiently?

Yes, it is possible to find the nearest point set associated with the Delaunay cell containing a given point $c \in R^d$. As we discussed in Section 3.2, the Delaunay complex can be represented by the convex hull of appropriately lifted points in R^{d+1}, and the non-vertical facets coincide with the Delaunay cells once they are projected to the original space. Thus the problem of determining the Delaunay cell containing a given point c can be reduced to finding the first facet of a polyhedron "shoot" by a ray.

To be more precise, let $f(x) = x_1^2 + \cdots + x_d^2$, and let $\tilde{p} = (p, f(x)) \in R^{d+1}$ for $p \in S$. Then the lower hull P of the lifted points $\tilde{S} := \{\tilde{p} : p \in S\}$:

$$P = conv(\tilde{S}) + nonneg(e^{d+1})$$

reprents the Delaunay complex. Here e^{d+1} is the unit vetor in R^{d+1} whose last component is 1. For any fixed vector $\tilde{y} \in R^{d+1}$ and $y_0 \in R$, let $\tilde{y}^T x \geq -y_0$ denote a general inequality of a variable vector $x \in R^{d+1}$. For such an inequality to represent a facet of P, it must be satisfied by all points in $\tilde{S}$:

$$\tilde{y}^T \tilde{p} \geq -y_0, \forall \tilde{p} \in \tilde{S},$$

and by any points shifted vertically upwards, that is,

$$\tilde{y}_{d+1} \geq 0.$$

Furthermore any non-vertical facet can be represented by such an inequality with $\tilde{y}_{d+1} = 1$. For a given point c, let $\tilde{c} = (c, 0)^T$, and let $L(\lambda) = \tilde{c} + \lambda\ e^{d+1}$, $\lambda \geq 0$. Now, determining the Delaunay cell containing c is equivalent to finding the first facet hit by the halfline L. In other words, it is to find a non-vertical facet inequality such that the intersecion point of the corresponding hyperplane $\{x : y^T x = -y_0\}$ and the half line $L(\lambda), \lambda \geq 0$ is highest possible.

By substituting $L(\lambda)$ for x in $y^T x = -y_0$ with $\tilde{y}_{d+1} = 1$, we obtain

$$\lambda = -y_0 - y^T c,$$

where y denotes the vector $\tilde{y}$ without the last coordinate $\tilde{y}_{d+1}$. The LP formulation is therefore:

$$\begin{array}{ll} \text{minimize } z := & y_0 + y^T c \\ \text{subject to} & f(p^i) + y_0 + y^T p \geq 0 \text{ for all } p \in S. \end{array} \tag{3}$$

While an optimal solution (y_0, y) to this LP may not determine any facet in general, the simplex method always returns an optimal basic solution which determines a facet inequality in this case. The Delaunay cell containing c is the one determined by the set of points in S whose corresponding inequalities are satisfied by equality at the optimal solution. If the LP solution is not degenerate, the dual variables that are positive at the dual optimal solution coincides with the former set.

It is important to note that the above LP might be unbounded. If it is unbounded, it can be easily shown that c in not in any Delaunay cell, i.e., not in the convex hull of S. A certificate of unboundedness actually induces a hyperplane strongly separating c from S. (why?)

3.6.1 Sample session with cdd+

With cdd+ and any reasonable LP code, the only necessary step should be to prepare the LP file for determination of the Delaunay cell containing a given point $c \in R^d$. Consider the same example 3.3.1.

For a given point $c = (3, 2)$, the LP (3) file for cdd+ is

```
H-representation
begin
  6  4  rational
  0  1  0  0
  5  1  2  1
  5  1  1  2
 16  1  4  0
 16  1  0  4
 32  1  4  4
end
minimize
  0  1  3  2
```

The solution by cddr+ is:

```
*LP status: a dual pair (x, y) of optimal solutions found.
begin
  primal_solution
  1 :  14
  2 :  -15/2
  3 :  -4
  dual_solution
```

```
  4 :  -1/8
  2 :  -1/2
  6 :  -3/8
  optimal_value : -33/2
end
```

Therefore, the facet inequality is $14 - 15/2x_1 - 4x_2 \geq 0$, and the dual solution indicate that the points p^2, p^4, p^6 determine the Delaunay cell which contains $(3, 2)$.

3.7 What is the best upper bound of the numbers of simplices in the Delaunay triangulation?

See 3.3.

4 Linear Programming

4.1 What is LP?

A linear programming (abbreviated by LP) is to find a maximizer or minimizer of a linear function subject to linear inequality constraints. More precisely,

$$\text{maximize} \quad f(x) := \sum_{j=1}^{d} c_j x_j \tag{4}$$

$$\text{subject to} \quad \sum_{j=1}^{d} a_{ij} x_j \leq b_i \text{ for all } i = 1, 2, \ldots, m, \tag{5}$$

where $A = [a_{ij}]$ is a given rational $m \times d$ matrix, $c = [c_j]$ and $b = [b_i]$ are given rational d- and n-vector. We often write an LP in matrix form:

$$\text{maximize} \quad f(x) := c^T x \tag{6}$$

$$\text{subject to} \quad A\, x \leq b. \tag{7}$$

LP is solvable in polynomial time by interior point methods and cab be efficiently solved by both the simplex method and interior-point methods. For example, it is easy on a standard unix station to solve an LP with $d = 100$ and $m = 100,000$, while the vertex enumeration/convex hull computation of the same size is simply intractable. There are many commercial codes and public codes available. See the LP FAQ [FG97]. Two excellent books on LP are Chvatal's textbook [Chv83] and Schrijver's "researcher's bible" [Sch86].

5 Polyhedral Computation Codes

- cdd and cdd+ [Fuk95] (C and C++ implementations of the double description method [MRTT53]).

  - floating and exact arithmetic, efficient for degenerate cases. The exact version cddr+ is much slower. It can remove redundancies from input data using a built-in LP code.

- lrs [Avi93] (C implementation of the reverse search algorithm [AF92]). A parallel version prs has been developed by A. Marzetta, see [BMFN96].

  - exact arithmetic only, efficient for nondegenerate cases. Uses a little memory and perhaps the only available code which can deal with problems generating huge output (say, one million vertices/facets).

- pd [Mar97] (C implementation of the primal-dual algorithm [BFM97]).

  - exact arithmetic only, efficient for dually nondegenerate cases.

- qhull [BDH95] (C implementation of the beneath-beyond method, see [Ede87, Mul94], which is the dual of the dd method).

  - floating arithmetic only, efficient for nondegenerate, and very fast for low dimensional cases. User can call it as a C-libary.

- Points to many other programs are stored in the Geometry Center software home page <http://www.geom.umn.edu/software/cglist>.

  (look for "convex hull").

References

[AF92] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.*, 8:295–313, 1992.

[Avi93] D. Avis. *A C implementation of the reverse search vertex enumeration algorithm.* School of Computer Science, McGill University, Montreal, Canada, 1993. programs lrs*.c available from ftp://mutt.cs.mcgill.ca/pub/C/.

[BDH95] C.B. Barber, D.P. Dobkin, and H. Huhdanpaa. *qhull, Version 2.1.* The Geometry Center, Minnesota, U.S.A., 1995. program and report available from ftp://geom.umn.edu/pub/software/qhull.tar.Z.

[BFM97] D. Bremner, K. Fukuda, and A. Marzetta. Primal-dual methods for vertex and facet enumeration. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 49–56, 1997.

[BMFN96] A. Brüngger, A. Marzetta, K. Fukuda, and J. Nievergelt. The parallel search bench zram and its applications. Technical report, ETH Zurich, May 1996. to appear in Annals of Operations Research, ps file available from ftp://ftp.ifor.math.ethz.ch/pub/fukuda/reports.

[Chv83] V. Chvatal. *Linear Programming.* W.H.Freeman and Company, 1983.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

[FG97] R. Fourer and J.W. Gregory. Linear programming frequently asked questions (LP-FAQ), 1997. http://www.mcs.anl.gov/home/otc/Guide/faq/linear-programming-faq.html.

[Fuk95] K. Fukuda. *cdd+ reference manual*. Institute for Operations Research, Swiss Federal Institute of Technology, Zurich, Switzerland, 1995. program available from http://www.ifor.math.ethz.ch/staff/fukuda/fukuda.html.

[Mar97] A. Marzetta. *pd – C-implementation of the primal-dual algoirithm*, 1997. code available from http://wwwjn.inf.ethz.ch/ambros/pd.html.

[MRTT53] T.S. Motzkin, H. Raiffa, GL. Thompson, and R.M. Thrall. The double description method. In H.W. Kuhn and A.W.Tucker, editors, *Contributions to theory of games, Vol. 2*. Princeton University Press, Princeton, RI, 1953.

[Mul94] K. Mulmuley. *Computational Geometry, An Introduction Through Randamized Algorithms*. Prentice-Hall, 1994.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, New York, 1986.

[Sei91] R. Seidel. Exact upper bounds for the number of faces in d-dimensional Voronoi diagram. In P. Gritzmann and B. Sturmfels, editors, *Applied Geometry and Discrete Mathematics - The Victor Klee Festschrift*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 517–529. Amer. Math. Soc., Providence, RI, 1991.