An Inductive constructive method for computation of the face lattice of a polyhedron

Doran K. Wilde, Sanjay Rajopadhye

An Inductive Constructive Method for Computation of the Face Lattice of a Polyhedron

Doran K. Wilde

Sanjay Rajopadhye

N˚ 2158

December 1993

PROGRAMME 1

Architectures parallèles,

bases de données,

réseaux et systèmes distribués

ISSN 0249-6399

An Inductive Constructive Method for Computation of the Face Lattice of a Polyhedron

Doran K. Wilde[*][**]
Sanjay Rajopadhye[***]

Programme 1 — Architectures parallèles, bases de données, réseaux et systèmes distribués
Projet API

Rapport de recherche n ° 2158 — December 1993 — 20 pages

Abstract: This article describes two algorithms to generate the face lattice of a polyhedron. A procedure by Motzkin which computes the dual of a polyhedron is presented and then extended to compute the entire face lattice of a polyhedron. This new algorithm recursively generates the lattice. Another nonrecursive algorithm to construct lattices which was invented by Seidel is also presented.

Key-words: Face Lattice, Polyhedra, Duality, Geometry

(Résumé : tsvp)

[*]The author may be contacted at `wilde@irisa.fr`
[**]This work was partially supported by the Esprit Basic Research Action NANA 2, Number 6632
[***]The author may be contacted at `rajopadh@irisa.fr`

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 38 38 32

Une Méthode Inductive pour le Calcul du Trellis des Faces d'un Polyèdre

Résumé : Cet article décrit deux algorithmes qui engendrent le treillis des faces d'un polyèdre. Une procédure de Motzkin qui calcule le dual d'un polyèdre est décrite et étendue afin de calculer le treillis des faces d'un polyèdre. Ce nouvel algorithme engendre récursivement le treillis. On décrit aussi un autre algorithme, non récursive, dû á Seidel pour résoudre le même problème.

Mots-clé : Treillis des Faces, Polyèdres, Dualité, Géometrie

1 Introduction

In this paper, a new algorithm to recursively construct the full face lattice of a polyhedron is presented. This is an extension of the work done in [Wil93]. In section 2, the basic definitions and properties relating to polyhedra are reviewed. This lays a foundation for the rest of the paper. In section 3, the fundamentals for the face lattice and duality are reviewed. In section 4, the Motzkin algorithm for computing the dual of a polyhedron is described in detail. Section 5 describes the algorithm of Seidel for computing the face lattice. In section 6, we extend the algorithm of Motzkin given in section 4 to produce a new algorithm to compute the face lattice. Section 7 shows an example and section 8 is a short summary.

2 Polyhedra and Faces of Polyhedra

This section is a review of fundamental definitions relating to polyhedra and cones. I have taken the majority of this summary from the works of Grunbaum, “Convex Polytopes” [Gru67], and of Schrijver, “Theory of Linear and Integer Programming” [Sch86], and of Edelsbrunner, “Algorithms in Combinatorial Geometry” [Ede87].

2.1 Notation and Prerequisites

In this presentation, polyhedra are restricted to being in the n-dimensional rational Cartesian space, represented by the symbol $\mathcal{Q}^n$. All matrices, vectors, and scalars are thus assumed to be rational unless otherwise specified.

Definition 1 *The* **scalar product** $a \circ b$ *is defined as* $a \circ b = a^T b = \sum_{i=1}^n a_i b_i$
where $a = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$ *and* $b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$.

2.2 The dual representations of polyhedra

Definition 2 *A* **polyhedron**, $\mathcal{P}$ *is a subspace of* $\mathcal{Q}^n$ *bounded by a finite number of hyperplanes.*
Alternate definition:
$\mathcal{P}$ *is the intersection of a finite family of closed linear halfspaces of the form* $\{x \mid ax \geq c\}$ *where* a *is a non-zero row vector and* c *is a scalar constant.*

A polyhedron $\mathcal{P}$ has a dual representation, an implicit and a parametric representation. The set of solution points which satisfy a mixed system of constraints form a polyhedron $\mathcal{P}$ and serve as the *implicit definition* of the polyhedron

$$\mathcal{P} = \{x \ : \ Ax = b, \ Cx \geq d\} \qquad (1)$$

given in terms of equations (rows of A, b) and inequalities (rows of C, d), where A, C are matrices, and b, d and x are vectors. This form corresponds to definition 2 above, where the set of closed halfspaces are defined by the inequalities: $Ax \geq b$, $Ax \leq b$, and $Cx \geq d$.

$\mathcal{P}$ has an equivalent dual *parametric representation* (also called the *Minkowski characterisation* after Minkowski— 1896 [Sch86, Page 87]) :

$$\mathcal{P} = \{x \ : \ x = L\lambda + R\mu + V\nu, \quad \mu, \nu \geq 0, \ \ \sum \nu = 1\} \qquad (2)$$

in terms of a linear combination of lines (columns of matrix L), a convex combination of vertices[1] (columns of matrix V), and a positive combination of extreme rays (columns of matrix R). The parametric representation shows that a polyhedron can be generated from a set of lines, rays, and vertices.

Procedures exist to compute the dual representations of $\mathcal{P}$, that is, given A, b, C, d, compute L, V, R, and visa versa. Such a procedure will be described later in section 4 of this paper.

2.3 The Polyhedral Cone

Polyhedral cones are a special case of polyhedra which have only a single vertex. (Without loss of generality, the vertex is at the origin.) A cone $\mathcal{C}$ is defined parametrically as :

$$\mathcal{C} = \{x \ : \ x = L\lambda + R\mu, \ \mu \geq 0\} \tag{3}$$

where L and R are matrices whose columns are the lines and extreme rays, respectively. If L is empty, then the cone is pointed.

Since the origin is always a solution point in Eq. 1, the implicit description of a cone has the following form

$$\mathcal{C} = \{x \ : \ Ax = 0, \ Cx \geq 0\} \tag{4}$$

the solution of a mixed system of *homogeneous* inequalities and equations.

2.4 Mapping from Inhomogeneous to Homogeneous Form

The transformation $x \longrightarrow \begin{pmatrix} \xi x \\ \xi \end{pmatrix}, \xi \geq 0$ changes an inhomogeneous system $\mathcal{P}$ of dimension n into a homogenous system $\mathcal{C}$ of dimension $n+1$. The original polyhedron $\mathcal{P}$ is in fact the intersection of the cone $\mathcal{C}$ with the hyperplane defined by the equality $\xi = 1$. Goldman showed that the mapping $x \longrightarrow \begin{pmatrix} \xi x \\ \xi \end{pmatrix}$ is one to one and inclusion preserving [Gol56] and thus this transformation does not change the face lattice.

Given any $\mathcal{P}$ as defined in equation 1, an unique homogeneous cone form exists defined as follows:

$$\begin{aligned} \mathcal{C} &= \{\hat{x} \mid \hat{A}\hat{x} = 0, \ \hat{C}\hat{x} \geq 0\} \\ &= \text{homogoneous.cone } \mathcal{P}, \\ \text{where } \hat{x} &= \begin{pmatrix} \xi x \\ \xi \end{pmatrix}, \ \hat{A} = \begin{pmatrix} A \mid -b \end{pmatrix}, \ \hat{C} = \left(\begin{array}{c|c} C & -d \\ \hline 0 & 1 \end{array}\right) \end{aligned} \tag{5}$$

2.5 Decomposition of the Cone

Following equation 3, a cone may be decomposed into [2] :

$$\mathcal{C} = \mathcal{L} + \mathcal{R} \tag{6}$$

the combination of the lineality space $\mathcal{L}$ (the linear combination of the lines of $\mathcal{C}$), and the ray space $\mathcal{R}$ (the positive combination of the extreme rays of $\mathcal{C}$). During the transformation process from a polyhedron to a cone, polyhedral vertices get transformed into rays in the cone. Rays in the polyhedron are also transformed to rays in the cone. Thus the ray space of the cone contains all of the vertices and rays of the original polyhedron.

[1] I am taking liberty with the term *vertices*. Here I use the term to mean the vertices of P less its lineality space.

[2] The symbol '+' in the equation is called the Minkowski sum, and is defined: $R + S = \{r + s \ : \ r \in R, \ s \in S\}$.

The dimensions of the lineality space and ray space are unique and separable since no irredundant ray is equal to a linear combination of lines (else the ray is redundant) and no line is a linear combination of rays (else the basis of ray space is redundant). Thus, the lineality space and ray space of a polyhedron are dimensionally distinct and the sum of their dimensions is the dimension of the polyhedron.

The structure of the face lattice is exclusively contained in the ray space of the cone. The lineality space has no effect on the lattice structure other than a 'dimensional displacement' of the entire lattice. Thus for computing the face lattice, the lineality space can be ignored.

2.6 Supporting Hyperplanes

Definition 3 *A hyperplane $\mathcal{H}$* **cuts** *a set $\mathcal{K}$ provided both open halfspaces determined by $\mathcal{H}$ contain points of $\mathcal{K}$, that is $\mathcal{H} = \{x \mid x \circ u = \alpha\}$* **cuts** *$\mathcal{K}$ iff there exists $x_1, x_2 \in \mathcal{K} \mid (x_1 \circ u < \alpha)$ and $(x_2 \circ u > \alpha)$*

Definition 4 *A* **supporting hyperplane** *is a plane which intersects the hull of a polyhedron, but does not cut $\mathcal{P}$, or in other words, does not intersect the interior of $\mathcal{P}$.*

Alternatively:

If c is a nonzero vector, and if $\delta = \max\{c \circ x \mid Ax \leq b\}$ exists, then the affine hyperplane $\mathcal{H} = \{x \mid c \circ x = \delta\}$ is a **supporting hyperplane** *of $\mathcal{P}$.*

The supporting hyperplane is a plane which just touches the surface of the polyhedron. The intersection of a supporting hyperplane and a polyhedron can be a point, edge, plane, or so forth.

2.7 Faces

Definition 5 *A subset $\mathcal{F}$ of $\mathcal{P}$ is called a* **face** *of $\mathcal{P}$ if either:*
(i) $\mathcal{F}$ is the intersection of $\mathcal{P}$ with a supporting hyperplane, or
(ii) $\mathcal{F} = \mathcal{P}$, or
(iii) $\mathcal{F} = \text{lineality.space}(\mathcal{P})$.

Case (iii), called the empty face, is added to force closure of the set of faces under intersection. Faces defined by cases (iii) and (ii) are called **improper faces** while faces defined by case (i) are called **proper faces**.

Every face of $\mathcal{P}$ is also a polyhedron and is called a **k-face** if it is a k-polyhedron. 0-faces are vertices. 1-faces are edges. The number of faces of a polyhedron is finite.

Definition 6 *The $(n-1)$-faces of a n-polyhedron are called* **facets** *and the 0-faces of a polyhedron are called vertices and rays. A* **facet** *of $\mathcal{P}$ is a maximal face distinct from $\mathcal{P}$ (maximal relative to inclusion). A* **minimal face** *of $\mathcal{P}$ is a nonempty face not containing any other nonempty face.*

Property 1 *Each minimal face of $\mathcal{P}$ is a translate of the lineality space of $\mathcal{P}$, and has the same dimension.*

Property 2 *The set of faces of a polyhedron form a lattice with respect to inclusion which is called the* **face lattice**.

Definition 7 *$f_k(\mathcal{P})$ is defined as the number of k-faces of polyhedron $\mathcal{P}$.*

Given a polyhedron $\mathcal{P} = \{x \mid Ax \geq 0, \quad Bx = 0\}$, there is a one-to-one correspondance between each nonredundant inequality $a_i x \geq 0$ that bounds the polyhedron and the corresponding facet $\mathcal{F}_i$ which is formed by intersecting the hyperplane $\mathcal{H}_i = \{x \mid a_i x = 0\}$ and $\mathcal{P}$ as stated in the following theorem:

Theorem 1 *(relating non redundant inequalities and facets)*
*There is a one to one correspondance between the facets of a polyhedron $\mathcal{P}$ and the irredundant inequalities of $\mathcal{P}$. Given $\mathcal{P} = \{x \mid Ax \geq 0,\ Bx = 0\}$, facet $\mathcal{F}_i = \{x \in \mathcal{P} \mid a_i x = 0\}$ (where a_i is the i*th *row of A) is in one-to-one correspondance with the inequality $a_i x \geq 0$. The constraint $a_i x \geq 0$* **defines** *or* **determines** *facet $\mathcal{F}_i$.*

Proof: Given a a_i, row i of matrix A, define hyperplane $\mathcal{H}_i = \{x \mid a_i x = 0\}$. $\mathcal{H}_i$ does not cut $\mathcal{P}$ since no $x \in \mathcal{P}$ exists such that $a_i x < 0$ (definition 3). Since a_i is a non-redundant row, some point in $\mathcal{H}$ is also in $\mathcal{P}$, thus $\mathcal{H}_i$ is a supporting hyperplane (definition 4) and $\mathcal{P} \cap \mathcal{H}_i$ is a face of $\mathcal{P}$ (definition 5). Calling that face $\mathcal{F}_i$ we have:

$$\begin{aligned}\mathcal{F}_i &= \mathcal{P} \cap \mathcal{H}_i \\ &= \{x \in \mathcal{P}\} \cap \{x \mid a_i x = 0\} \\ &= \{x \in \mathcal{P} \mid a_i x = 0\} \ \cdot\end{aligned}$$

Since $\mathcal{F}_i$ is the result of the intersection of $\mathcal{P}$ with a non-redundant equality, the face is a polyhedron of dimension one less than $\mathcal{P}$ and is thus a facet of $\mathcal{P}$ (definition 6).

□

Assuming A does not contain any redundant constraints (rows), the number of facets is equal to the number of rows in A since each *facet* of $\mathcal{P}$ is *defined* or *determined* by a unique row of A.

In general, any face $\mathcal{F}$ of polyhedron $\mathcal{P}$ can be determined by a unique subset of rows of A. For each face $\mathcal{F}$, there exists a row submatrix A' of A, such that $\mathcal{F}$ can be described as:

$$\mathcal{F} = \{x \in \mathcal{P} \mid A'x = 0\} \ \cdot$$

3 The Face-Lattice

The relation $f \vdash g$, "f is a subface of g", is transitive and anti-symmetric and hence can be used to define a partial order among the faces of a polyhedron.

Property 3 *(Transitive Property of the $\vdash$ relation)*
If $f \vdash g$ and $g \vdash h$ then $f \vdash h$.

Property 4 *(Anti-symmetry property of the $\vdash$ relation)*
If $f \vdash g$ and $f \neq g$ then $g \not\vdash f$.

The $\vdash$ relation, along with the partially ordered set of all of the faces of a polyhedron (definition 5), form a lattice called the **face lattice** with the n-dimensional polyhedron at the top, and the empty face (called the -1-face) at the bottom (figure 1).

This lattice induces a directed graph called the **facial graph** in which the nodes are the faces of $\mathcal{P}$ and a directed edge exists between nodes f and g if and only if g is a facet of f. The size of the facial graph of $\mathcal{P}$ is the number of nodes and arcs and is denoted by $L(\mathcal{P})$. The number of vertices and extremal rays of a polyhedron (the 0-faces) is written as $f_0(\mathcal{P})$. Furthermore, since there is a one to one correspondance between the non-redundant constraints in the implicit description of a polyhedron and the facets of that polyhedron (theorem 1), the number of non-redundant constraints which is the number of facets can be written as $f_{d-1}(\mathcal{P})$. It has been shown that for a d-polyhedron $\mathcal{P}$ that both $L(\mathcal{P})$ and the number of vertices and rays $f_0(\mathcal{P})$ are $\mathcal{O}(k^{\lfloor \frac{d}{2} \rfloor})$ where $k = f_{d-1}(\mathcal{P})$, the number of constraints [Ede87].

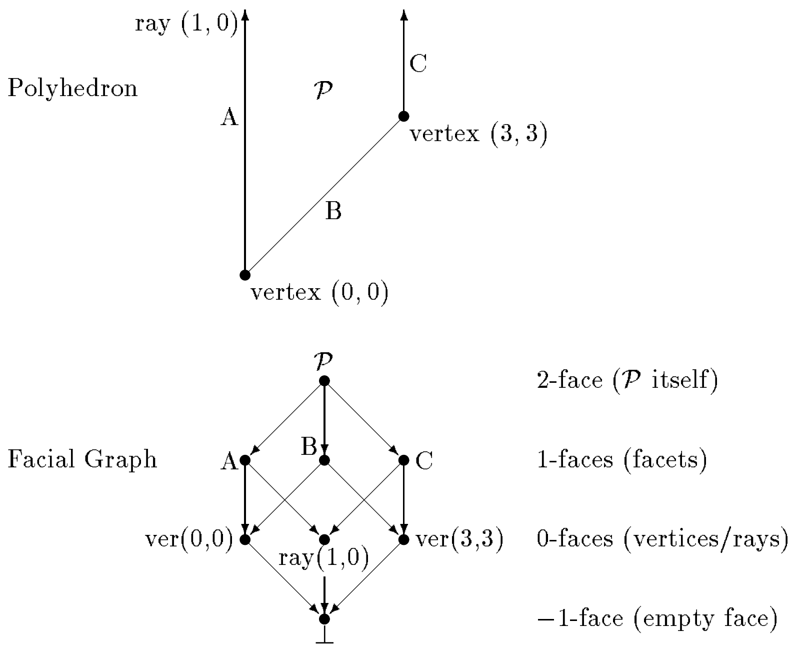

Figure 1: Example of a Facial Graph

3.1 Lattices of dual polyhedra

Definition 8 *Two d-polytopes, $\mathcal{P}$ and $\mathcal{P}*$ are said to be* **dual** *to each other provided there exists a 1-1 mapping between the set F of all faces of $\mathcal{P}$, and the set $F*$ of all faces of $\mathcal{P}*$, such that the mapping is inclusion-reversing. in other words, F_1 is a face of F_2 iff map(F_2) is a face of map(F_1).*

Definition 9 *(Polar)*
Given a closed convex set P containing the point 0, then the polar *$P*$ is defined as*
$P* = \{y \mid \forall x \in \mathcal{P} \ : \ x \circ y \geq 0\}$.

Property 5 *(duality of polars)*
If $P*$ is the polar of P, then P and $P*$ are duals of each other.

We can show the duality of a system of constraints with its corresponding system of lines and rays. Let $\mathcal{C}$ be a cone and $\mathcal{C}*$ be another cone created by reinterpreting the inequalities and equalities of $\mathcal{C}$ as the lines and rays, respectively, of $\mathcal{C}*$. Then the two cones are defined as:

$$\begin{aligned} \mathcal{C} &= \{x \mid x = L\lambda + R\mu, \ \mu \geq 0\} \ = \ \{x \mid Ax = 0, \ Cx \geq 0\} \\ \mathcal{C}* &= \{y \mid y = A^T\alpha + C^T\gamma, \ \gamma \geq 0\} \ = \ \{y \mid L^T y = 0, \ R^T y \geq 0\} \end{aligned}$$

The inner product of a point $x \in \mathcal{C}$ and a point $y \in \mathcal{C}*$ can be shown to be $x \circ y \geq 0$ [Wil93] from which follows

$$\mathcal{C}* = \{y \mid \forall x \in \mathcal{C} \ : \ x \circ y \geq 0\}$$

and thus $\mathcal{C}$ and $\mathcal{C}*$ are duals by property 5.

The definition of dual polyhedra (definition 8) states that two polyhedra are dual to each other when there is a 1-1 mapping from faces of one to the faces of the other which is inclusion reversing. Let M be such a mapping between polyhedra $\mathcal{P}$ and $\mathcal{Q}$, then

(i) for each face f in $\mathcal{P}$, $M(f)$ is a face of $\mathcal{Q}$.

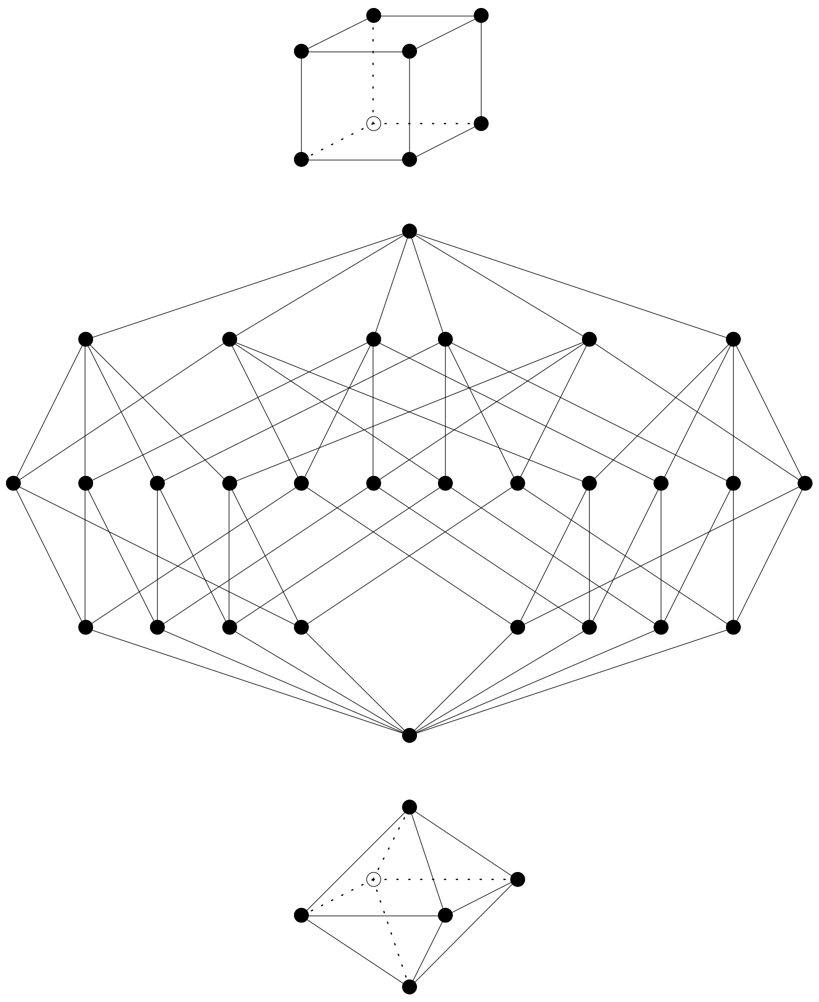

Figure 2: Face Lattice of Dual Polyhedra

(ii) for each incidence $f \vdash g$ in $\mathcal{P}$, $M(g) \vdash M(f)$ is an incidence in $\mathcal{Q}$.

This implies that the face lattice of $\mathcal{P}$ and $\mathcal{Q}$ are exact inversions of each other. Figure 2 shows two such polyhedra. The face graph interpreted from top to bottom represents the face lattice of the polyhedron on the top. The face graph interpreted from bottom to top represents the polyhedron on the bottom. There is a 1-1 correspondance between facets of one polyhedron and vertices of the other, and visa versa. The reason that duality is important is that (for instance) everything proven for facets, by duality, is proven for rays. Duality can be looked at from the point of view of two dual polyhedra, or from the point of view of the dual representation (constraint representation vs. ray representation) of a single polyhedron. We make the most advantage of the second point of view.

4 Computation of Dual Forms

An important problem in computing with polyhedral domains is being able to convert from a domain described implicitly in terms of linear equalities and inequalities (equation 1), to a parametric description (equation 2) given in terms of the geometric features of the polyhedron (lines, rays, and vertices). Inequalities and equalities are referred to collectively as *constraints*. An equivalent problem is called the *convex hull problem* which computes the facets of the convex hull surrounding given a set of points.

McMullen [McM70, MS71] showed that for any d-polytope with n vertices, the number of k-faces, f_k is upper bounded by the number of k-faces of a cyclic d-polytope with the same number of vertices. One of the implications of this is that the number of facets, $f_{d-1} = \mathcal{O}(n^{\lfloor \frac{d}{2} \rfloor})$.

The algorithms to solve this problem are categorized into one of two general classes of algorithms, the pivoting and non-pivoting methods [MR80]. The pivoting methods are derivatives of the simplex method which finds new vertices located adjacent to known vertices using simplex pivot operations.

The nonpivoting methods find the dual by first setting up a tableau in which an initial polyhedron (such as the universe or the positive orthant) is simultaneously represented in both forms. The algorithm then iterates by adding one new inequality or equality at a time and computing the new polyhedron at each step by modifying the polyhedron from the previous step. The order in which constraints are selected does not change the final solution, but may have an effect on the run time of the procedure as a whole. The complexity of this problem is known to be $\mathcal{O}(n^{\lfloor \frac{d}{2} \rfloor})$, where n is the number of constraints and d is the dimension. This is the best that can be done, since the size of the output (i.e. the number of rays) is of the same order.

The nonpivoting methods are based on an algorithm called *the double description method* invented by Motzkin et al. in 1953 [MRTT53]. Motzkin described a general algorithm which iteratively solves the dual-computation problem for a cone. (Since polyhedra may be converted to cones, it works for all polyhedra.) In each iteration, a new constraint is added to the current cone in the tableau. Rays in the cone are divided into three groups, R^+ the rays which verify the constraint, R^0 the rays which saturate the constraint, and R^- the rays which do not verify the constraint. A new cone is then constructed from the ray sets R^+, R^0, plus the convex combinations of pairs of rays, one each from sets R^+ and R^-. The main problem with the nonpivoting methods is that they can generate a non-minimal set of rays by creating non-extreme or redundant rays. If allowed to stay, the number of rays would grow exponentially and would seriously test the memory capacity of the hardware as well as degrade the performance of the procedure. Motzkin proposed a simple and rather elegant test to solve this problem. He showed that a convex combination of a pair of rays ($r^- \in R^-$, $r^+ \in R^+$) will result in an extreme ray in the new cone if and only if the minimum face which contains them both: 1) is dimension one greater than r^- and r^+, and 2) only contains the two rays r^- and r^+. This test inhibits the production of unwanted rays and keeps the solution in a minimal form.

Seidel described an algorithm for the equivalent convex hull problem [Sei91] which executes in $\mathcal{O}(n^{\lfloor \frac{d}{2} \rfloor})$ expected running time where n is the number of points and d is the dimension. This is provably the best one can do, since the output of the procedure is of the same order. He solves the adjacent ray problem (the adjacent facet problem in his case) by creating and maintaining a facet graph in which facets are vertices and adjacent facets are connected by edges. It takes a little extra code to maintain the graph, but then he does not need to do the Motzkin adjacency test on all pairs of vertices (facets).

4.1 The Motzkin algorithm

The nonpivoting solvers successively refine their solution by adding one constraint at a time and modifying the solution polyhedron from the previous step to reflect the new constraint. An inequality $a^T x \geq 0$ is co-represented by the closed halfspace H^+ which is the set of points $\{x \ : \ a^T x \geq 0\}$.

Likewise the equality $a^T x = 0$ is co-represented by the hyperplane H which is the set of points $\{x : a^T x = 0\}$. At each step of the algorithm, a new inequality or equality (represented by either H^+ or H, respectively) is introduced into the system. The polyhedron $\mathcal{P} = \mathcal{L} + \mathcal{R}$ (the combination of its lineality space and ray space) is constrained by the new constraint by intersecting $\mathcal{P}$ with either H^+ or H to produce a modified polyhedron $\mathcal{P}' = \mathcal{L}' + \mathcal{R}'$.

The algorithm **Dual** in figure 3 gives the algorithm given by Motzkin to find the dual of a set of constraints A. In **Dual**, there are three procedures which alter the polyhedron. They are **ConstrainL** which constrains the lineality space, **AugmentR** which augments the dimension of the ray space, and finally **ConstrainR** which constrains the ray space. These procedures are discussed below in greater detail.

The **ConstrainL** procedure shown in figure 5 constrains the lineality space L by slicing it with a new constraint, and if the new constraint cuts L, then L's dimension is reduced by one and a new ray r_{new} is generated which is added to the ray space. It is fairly straightforward and runs in $\mathcal{O}(n)$ time where n is the dimension of the lineality space.

There are two procedures which perform transformations on the ray space. The first one is **AugmentR** shown in figure 6 which adds a new ray r_{new} created by **ConstrainL** to the ray space. When r_{new} is added to the ray space R, it increases the dimension of R by one. It is of complexity $\mathcal{O}(r)$ time, where r is the number of rays.

The second operation **ConstrainR** shown in figure 7 constrains the ray space by slicing it with the hyperplane H and discarding the part of the ray space which lies outside of constraint. For inequalities, the part of the polyhedron which lies outside of the halfspace H^+ is removed. For equalities, the part of the polyhedron which lies outside of the hyperplane H is removed. In either case, the new face lying on the cutting hyperplane surface is computed. **ConstrainR** computes a new pointed cone by adding a new constraint. Rays which verify and saturate the constraint are added. Rays which do not verify the constraint are combined with *adjacent* rays which verify the constraint to create new rays which saturate the constraint. Motzkins adjacency test is used to find adjacent pairs of rays. The Motzkin adjacency test is used to test every pair of rays to determine if that pair will combine to produce an extreme ray or not. This is done by computing what constraints the pair of rays have in common and making sure that no other ray also saturates that same set of constraints. Thus the list of rays produced by **ConstrainR** is always extreme (non-redundant). The entire **ConstrainR** procedure has an $\mathcal{O}(n^3 k)$ complexity where n is the number of rays and k is the number of constraints. Much of this time is spent in performing the adjacency tests.

The procedure **Combine** shown in figure 4 is where all of the actual computation takes place. It uses as input two rays, r^+ and r^-, as well as a constraint a. It then computes the ray $r^=$ which firstly is a linear combination of r^+ and r^- ($r^= = \lambda_1 r^+ + \lambda_2 r^-$), and secondly, saturates constraint a, ($a^T r^= = 0$).

```
Procedure **Dual**($A$), returns $L$, $R$

$L$ := basis for $d$-dimensional lineality space.
$R$ := point at the origin.
For each constraint $a \in A$ Do
    $r_{new}$ := **ConstrainL**($L$, $a$)
    If $r_{new} \neq 0$ Then **AugmentR**($R$, $a$, $r_{new}$) Else **ConstrainR** ($R$, $a$)
End
Return $L$ and $R$.
```

Figure 3: Procedure to compute **Dual**(A)

```
Procedure **Combine**($r_1$, $r_2$, $a$), returns $r_3$

$D = \text{GCD}(a^T r_1, a^T r_2)$
$\lambda_1 = a^T r_2 / D$
$\lambda_2 = -a^T r_1 / D$
$r_3 = \lambda_1 r_1 + \lambda_2 r_2$
```

Figure 4: Procedure to compute **Combine**(r_1, r_2, a)

```
Procedure **ConstrainL**($L$, $a$), modifies $L$, returns $r_{new}$

Find an $l_1 \in L$ such that $a^T l_1 \neq 0$, ($l_1$ does not saturate constraint $a$)
If $l_1$ does not exist Then ($L \cap H$ is $L$ itself and $r_{new}$ is empty) Return 0.
$L'$ := empty.
For each line $l_2 \in L$ such that $l_2 \neq l_1$ Do
    $L' := L' +$ **Combine**($l_1$, $l_2$, $a$)
End
If $a^T l_1 > 0$, ($l_1$ verifies constraint $a$) Then Create ray $r_{new}$ equal to $l_1$
Else ($a^T l_1 < 0$) Create ray $r_{new}$ equal to $-l_1$
$L := L'$
Return $r_{new}$
```

Figure 5: Procedure to compute **ConstrainL**(L, a)

```
Procedure **AugmentR**($R$, $a$, $r_{new}$), modifies $R$

Set $R'$ := empty.
For each ray $r \in R$ do
    If $a^T r = 0$ Then $R' := R' + r$
    If $a^T r > 0$ Then $R' := R' +$ **Combine**($r$, $-r_{new}$, $a$)
    If $a^T r < 0$ Then $R' := R' +$ **Combine**($r$, $r_{new}$, $a$)
End
If $a$ is an inequality Then $R' := R' + r_{new}$
$R := R'$
```

Figure 6: Procedure to compute **AugmentR**(R, a, r_{new})

```
Procedure **ConstrainR**($R$, $a$), modifies $R$

Partition $R = R^+ + R^0 + R^-$.
    $R^0 := \{r \ : \ r \in R, a^T r = 0\}$, the rays which saturate constraint $a$.
    $R^+ := \{r \ : \ r \in R, a^T r > 0\}$, the rays which verify constraint $a$.
    $R^- := \{r \ : \ r \in R, a^T r < 0\}$, the rays which do not verify constraint $a$.
If constraint $a$ is an inequality, Then set $R' := R^+ + R^0$.
Else (constraint $a$ is an equality) set $R' := R^0$.
For each ray $r^+ \in R^+$ do
    For each ray $r^- \in R^-$ do
        **Adjacency test on** $(r^+, r^-)$
        c := set of common constraints saturated by both $(r^+, r^-)$
        For each ray $r \in R \ : \ r \neq r^+, \ \ r \neq r^-$ Do
            If $r$ also saturates all of the contraints in set $c$ Then
                ($r^+$ and $r^-$ are not adjacent.) Continue to next ray $r^-$.
        End
        ( $r^+$ and $r^-$ are adjacent.) $R' := R' +$ **Combine**($r^+$, $r^-$, $a$)
    End
End
$R := R'$
```

Figure 7: Procedure to compute **ConstrainR**(R, a)

5 Previous Art in Lattice Construction

Grunbaum stated the basic theorem which, given a set of vertices, generates all faces of the minimal polytope containing those vertices. He did this in an iterative fashion, adding one vertex at a time to an existing polyhedron $\mathcal{P}$, and computing the faces of the new polyhedron $\mathcal{P}*$ given the new point and the faces of the old polyhedron $\mathcal{P}$. This generates the faces of the facial graph but not the arcs (incidences) between the faces.

Before giving the theorem, three definitions are needed. Letting P be a n-polytope, H be a hyperplane such that H does not cut P, and V be a point, then the following definitions are given:

Definition 10 *V is* beneath *H (with respect to P) provided V belongs to the open halfspace determined by H which contains* internal *P. [inside]*

Definition 11 *V is* beyond *H (with respect to P) provided V belongs to the open halfspace determined by H which does not meet P. [outside]*

Definition 12 *V is* on *H provided V belongs to the hyperplane H. [on]*

The relation between the set of faces of a polytope P and that of the convex hull of P plus one additional point V is given by the following theorem:

Theorem 2 *(Theorem by Grunbaum)*
Let P and $P*$ be two n-polytopes in $\mathcal{Q}^n$, and let V be a vertex of $P*$ but not of P, such that $P* = \text{convex.hull}(P \cup \{V\})$. Then,

(i) *a face F of P is also a face of $P*$ iff there exists a facet F' of P such that F in F' and V is beneath F';*

(ii) *if F is a face of P then $F* = \text{conv}(F \cup \{V\})$ is a face of $P*$ iff either*

  (a) *V is in* affine.hull *F, or*

  (b) *among the facets of P containing F, there is at least one such that V is beneath it and at least one such that V is beyond it.*

(iii) *each face of P* is generated by either rule (i) or (ii) above.*

5.1 Seidel's method

Seidel added the generation of incidences to the procedure of Grunbaum, and thus was able to generate the full face lattice (faces and incidences) of a polytope surrounding a given set of points. Like the Grunbaum procedure, the Seidel procedure adds one point p at a time to P to get P', iteratively building up the lattice. The procedure **AddPoint** which updates a list of faces and incidences, given a new point, is presented on the next page.

When adding a new point to an existing polytope, the algorithm differentiates two cases:
(1) the point is not in the affine hull of P, and
(2) the point is in the affine hull of P.
In the first case, the polytope will grow a dimension. In the second case, the dimension of the polytope stays the same.

```
Procedure **AddPoint**$(P, p)$, Returns $P' = \text{convex.hull}(P \cup p)$

If $p$ not in affine.hull$(P)$ Then
    ($P'$ is created in which dim$(P')$ = dim$(P)$+1).
    For every face $f$ of $P$ Do
        (Find faces of $P'$ when $p$ is not in affine.hull$(P)$)
        $f' = f$ is a face of $P'$
        $f'' = \text{convex.hull}(f \cup \{p\})$ is a face of $P'$
    For pairs of faces $f$ and $g$ of $P$ Do
        (Find incidences of $P'$ when $p$ is not in affine.hull$(P)$)
        Let $f'$,$f''$,$g'$,$g''$ be faces of $P'$ induced by faces $f$ and $g$ in $P$
        $f' \vdash g'$ in $P'$ iff $f \vdash g$ in $P$.
        $f'' \vdash g''$ in $P'$ iff $f \vdash g$ in $P$.
        $f' \vdash f''$ in $P'$ iff $f$ in $P$.
Else ($p$ is in affine.hull$(P)$)
    ($P'$ is created in which dim$(P')$ = dim$(P)$).
    **Classification of facets of $P$**
        For each facet $f$ of $P$ Do
            Let hyperplane $h$ = affine.hull$(f)$.
            $f$ is [out] if $p$ is beyond $h$.
            $f$ is [on] if $p$ is contained in $h$.
            $f$ is [in] if $p$ is beneath $h$.
        End
    **Classification of other faces of $P$**
        For each $k$-face $e$ of $P$ which is not a facet ($k <$dim$(P) - 1$) Do
            $e$ is [out,on] if $e$ is bounded by [out] and [on] faces.
            $e$ is [in,on] if $e$ is bounded by [in] and [on] faces.
            $e$ is [in,out] if $e$ is bounded by [in] and [out] faces.
            $e$ is [in,on,out] if $e$ is bounded by [in], [on], and [out] faces.
        End
    For every face $f$ of $P$ Do
        (Find faces of $P'$ when $p$ is in affine.hull$(P)$)
        $f' = f$ is a face of $P'$ if $f$ has a [in] component.
        $f''$ =convex.hull$(f \cup \{p\})$ is a face of $P'$ if $f$ has [in] and [out] components.
        $f'''$ =convex.hull$(f \cup \{p\})$ is a face of $P'$ if $f$ is [on] or if $f = P$
    For all pairs of faces $f$ and $g$ of $P$ Do
        (Find incidences of $P'$ when $p$ is in affine.hull$(P)$)
        Let $f'$, $f''$, $f'''$, $g'$, $g''$, $g'''$ be faces induced in $P'$ by faces $f$ and $g$ in $P$
        $f' \vdash g'$ iff $f \vdash g$
        $f' \vdash g''$ iff $f = g$
        $f' \vdash g'''$ iff $f \vdash g$
        $f'' \vdash g''$ iff $f \vdash g$
        $f'' \vdash g'''$ iff there exists subface $x$ of $g$ where $f \vdash x \vdash g$
        $f''' \vdash g'''$ iff $f \vdash g$
    End
End
```

Figure 8: Seidel's algorithm

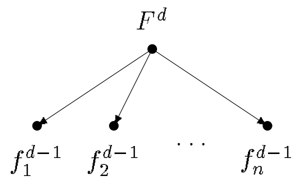

Figure 9: Facial Graphs for a Face and its Facets

6 The Inductive Face Lattice Algorithm

A modification of Motzkin's **Dual** algorithm (presented in section 4.1) can be used to produce the entire face lattice of a polyhedron using an inductive constructive method.

The algorithm presented here constructs the face lattice of a polyhedron from a list of constraints. A polyhedron P is stored as a homogenous cone represented by the union of the lineality space L and the ray space R with an added data structure to represent the lattice. The lattice is represented by a hierarchy of faces with incident faces connected with pointers. The top of the face lattice is the whole polyhedron and at the bottom are the extreme rays of R. (The empty face is not represented.) Each face in the lattice has a dimension corresponding to its level in the lattice. Thus, a lattice for a d-dimensional ray space would have $d+1$ levels, the levels having dimensions $d, d-1, \cdots, 0$, from top to bottom.

6.1 Data structure for face lattice

A k-face is composed of a set of $(k-1)$-facets (subfaces of dimension $k-1$) as shown in figure 9.

$$F^k = \{f_0^{k-1}, f_1^{k-1}, \cdots, f_n^{k-1}\} \quad .$$

A facial graph, a piece of which is shown in figure 9, consists of nodes representing faces and edges representing incidences from a face to its facets. Each face is a node in the face lattice and is represented by a data structure with the following fields :

dimension The dimension of this face.

flags A set of face attributes in the context of the current constraint. Attributes defined are:
in: set when this face verifies the constraint,
on: set when this face saturates the constraint,
out: set when this face does not verify the constraint.
A face can have any non-empty combination of these attributes in its flag set.

facets The set of facets (of dimension one less than the dimension of this face) that are incident to this face. This field is the head of a linked list of pointers to subface nodes in the face lattice (figure 9).

ray (Only used for dimension 0 faces). The extreme ray in R corresponding to this face.

Using a standard "dot notation", these fields will be referred to as F.dim, F.flags, F.facets, F.ray in the remainder of this paper.

6.2 Modifications to the Dual procedure

The main procedure for creating the face lattice of a polyhedron is the same as the procedure **Dual** described in section 4.1 and shown in figure 3. **Dual** calls three procedures: **ConstrainL**,

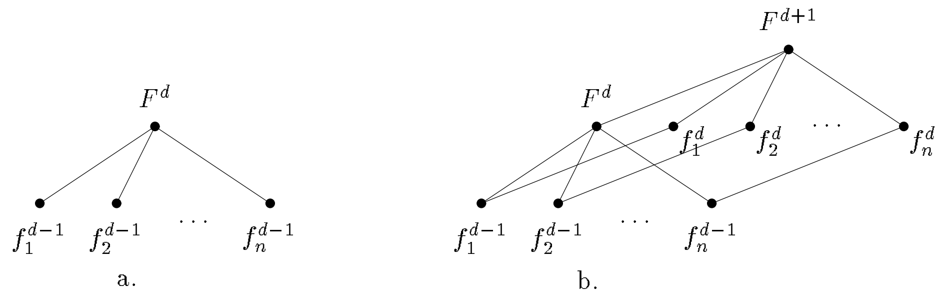

Figure 10: Facial Graphs for **AugmentR** Algorithm

AugmentR and **ConstrainR**. The procedure **ConstrainL**, which is used to constrain the lineality space and is shown in figure 5, does not need to change from the one previously described in section 4.1 to compute the face lattice since the lineality space does not directly affect the face lattice. The differences in between finding the dual of a polyhedron and finding the lattice of are polyhedron are found in the computation on the ray space R by procedures **AugmentR** and **ConstrainR**. These two subprocedures, which determine the way that the ray space R is constrained and augmented, have been rewritten in order to generate the entire lattice. The procedures **AugmentR** and **ConstrainF** (which is called by **ConstrainR**) recursively construct the lattice are the primary contributions of this paper. The procedure **AugmentR** adds a new basis ray r_{new} which is not in the affine hull of R to the lattice— increasing the dimension of the lattice by one. The procedure **ConstrainF** recursively constrains the lattice by slicing it with a new constraint. The resulting lattice is of the same dimension, however parts of the lattice outside the new constraint are removed and replaced with the new face created by the cut.

6.3 AugmentR

The procedure **AugmentR** shown if figure 11 adds a ray r_{new} which is not in the affine hull of R to the lattice F^d representing a face of dimension d, and returns an augmented face F^{d+1} of dimension $d+1$. It is computed recursively as follows:

$$\begin{array}{ll}
\text{Given} & F^d = \{\ f_1^{d-1},\ f_2^{d-1},\ \cdots,\ f_n^{d-1}\} \\
\text{then} & F^{d+1} = \text{convex.hull}(F^d \cup \{r_{new}\}) = \{F^d,\ f_1^d,\ f_2^d,\ \cdots,\ f_n^d\} \\
\text{where} & f_1^d = \text{convex.hull}(f_1^{d-1} \cup r_{new}) \\
 & f_2^d = \text{convex.hull}(f_1^{d-1} \cup r_{new}) \\
 & \vdots \\
 & f_n^d = \text{convex.hull}(f_1^{d-1} \cup r_{new}).
\end{array}$$

Figure 10a. shows a piece of the Hasse diagram of the lattice F^d. Figure 10b. shows the same piece of the diagram after it has been augmented by adding a new basis ray using this procedure. A new face F^{d+1} is created and assigned the following subfaces: (1) F^d (the original face) and (2) the new faces $f_1^d, f_2^d, \cdots, f_n^d$ which are computed by recursively calling this procedure on $f_1^{d-1},\ \ f_2^{d-1}, \cdots, f_n^{d-1}$, respectively (the facets of the original face F^d).

6.4 ConstrainR

The **ConstrainR** procedure is detailed in figure 13. First of all, the recursive procedure **Evaluate** (shown in figure 14) is called which evaluates constraint a on each of the faces of polyhedron R and marks status flags in each face, saying whether each face verifies, saturates, or does not verify the constraint, or a combination of the three for non trivial faces. After evaluating the faces in light of the constraint, **ConstrainR** either returns an empty lattice if none of the polyhedron verifies or saturates the constraint, or returns the saturating faces if only a part of the polyhedron saturates

```
**AugmentR**($F$, $a$, $r_{new}$), Modifies $F$, returns $F_{new}$ =convex.hull($F \cup r_{new}$)

If $F_{new}$ =**AugmentR**($F$, $a$, $r_{new}$) has already been computed
 Then return $F_{new}$
$F_{new}$ := null
If $F_{.\text{dim}}$ == 0 Then (face $F$ is a ray)
    If $a^T F_{.\text{ray}} \neq 0$ Then (constraint not saturated)
        Allocate a new face $r_1$
        If $a^T F_{.\text{ray}} > 0$ $r_{1.\text{ray}}$ =**Combine**($F_{.\text{ray}}, -r_{new\,.\text{ray}}, a$)
        Else ($a^T F_{.\text{ray}} < 0$) $r_{1.\text{ray}}$ =**Combine**($F_{.\text{ray}}, r_{new\,.\text{ray}}, a$)
        If constraint $a$ is an inequality Then Create $F_{new}$ with subfaces $\{F,\ r_1\}$
Else ( $F_{.\text{dim}} > 0$ )
    For each facet $f$ of face $F$ Do
        $g$=**AugmentR**($f$, $a$, $r_{new}$)     (recursive call)
        If $g$ is not null
            If $F_{new}$ is null
                Create new face $F_{new}$ (with no facets)
                Add $F$ as a facet to augmented face $F_{new}$
            Add face $g$ as a facet to augmented face $F_{new}$
    End
Return augmented face $F_{new}$
```

Figure 11: Procedure to compute **AugmentR**(F, a, r_{new})

the constraint, or returns the constrained lattice if the polyhedron partially verifies and partially does not verify the constraint. The last two cases are done by calling the **SelectF** and **ConstrainF** procedures respectively.

6.4.1 Evaluate

The procedure **Evaluate** (shown in figure 14) to evalute a constraint a on a face lattice F is also a recursive procedure, setting the flags of a face F as follows:

$$F_{.\text{flags}} = \begin{cases} \bigcup_i^n (f_i)_{.\text{flags}} & F_{.\text{dim}} > 0,\ F = \{f_1, f_2, \cdots, f_n\} \\ \{\text{in}\} & F_{.\text{dim}} = 0 \text{ and } a^T F_{.\text{ray}} > 0 \\ \{\text{on}\} & F_{.\text{dim}} = 0 \text{ and } a^T F_{.\text{ray}} = 0 \\ \{\text{out}\} & F_{.\text{dim}} = 0 \text{ and } a^T F_{.\text{ray}} < 0 \end{cases} .$$

The procedure performs a depth first traversal of the face lattice.

6.4.2 SelectF

The procedure **SelectF** is simply a breadth first search of the face lattice looking for the highest dimensioned face in the lattice which entirely saturates the constraint a, or in other words, the highest dimensioned face with $F_{.\text{flags}}$ ={on}. This procedure is straightforward and its details are not presented here.

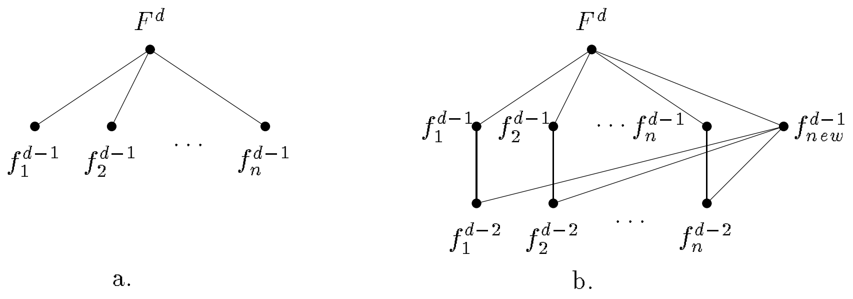
F^d
f_1^{d-1} f_2^{d-1} … f_n^{d-1}
a.
F^d
f_1^{d-1} f_2^{d-1} … f_n^{d-1} f_{new}^{d-1}
f_1^{d-2} f_2^{d-2} … f_n^{d-2}
b.

Figure 12: Facial Graphs for **ConstrainF** Algorithm

6.4.3 ConstrainF

The procedure **ConstrainF** (shown in figure 15 constrains the lattice F by cutting it with the constraint a represented by the halfspace $H^+ = \{x \mid a^T x \geq 0\}$. The the resulting lattice will be of the same dimension, however the parts of the lattice outside of the slicing hyperplane will be removed and a new face created by the cut plane will be added. The procedure builds the new face f_{new} which consists of the new facets of dimension $(d-2)$ created by cutting each of the original facets of F^d: $f_1^{d-1},\ f_2^{d-1}, \cdots, f_n^{d-1}$ and which lies on the cutting hyperplane. The procedure then links f_{new} as a new subface to F, and then returns the new face f_{new} to the caller. When procedure **ConstrainF** is called, face F^d is modified to be $F^d \cap H^+$, which is computed as follows:

$$
\begin{array}{ll}
\text{Given} & F^d = \{f_1^{d-1}, f_2^{d-1}, \cdots, f_n^{d-1}\} \\
& H^+ = \{x \mid a^T x \geq 0\}, \quad H^0 = \{x \mid a^T x = 0\} \\
\text{then} & F^d \cap H^+ = \{f_1^{d-1} \cap H^+, f_2^{d-1} \cap H^+, \cdots, f_n^{d-1} \cap H^+, f_{new}^{d-1}\} \\
\text{where} & f_{new}^{d-1} = F^d \cap H^0 = \{f_1^{d-2},\ f_2^{d-2}, \cdots, f_n^{d-2}\} \\
& f_1^{d-2} = f_1^{d-1} \cap H^0 \\
& f_2^{d-2} = f_2^{d-1} \cap H^0 \\
& \vdots \\
& f_n^{d-2} = f_n^{d-1} \cap H^0
\end{array}
$$

Figure 12a. shows a piece of the Hasse diagram of the lattice F^d. Figure 12b. shows the same piece of the diagram after it has been constrained by this procedure.

```
ConstrainR($R$,$a$) Modifies $R$

Call Evaluate($R$, $a$)
If $R_{.\text{flags}}$ = { in, on } or { in } or { on } then no change.
Else if $R_{.\text{flags}}$ = { out } then set $R$ := null ray space.
Else if $R_{.\text{flags}}$ = { on, out } then set $R$ := SelectF($R$, $a$)
Else ($R_{.\text{flags}}$ = { in, out } or { in, out, on }) set $R$ := ConstrainF($R$, $a$)
```

Figure 13: Procedure for computing **ConstrainR(**R,a**)**

```
**Evaluate**(F, a) Modifies the flags in $F$

If $F_{.\text{dim}} = 0$ then
    $x := a^T F_{.\text{ray}}$
    If $x = 0$ then $F_{.\text{flags}} := \{\text{on}\}$.
    Else if $x > 0$ then $F_{.\text{flags}} := \{\text{in}\}$.
    Else $(x < 0)$ $F_{.\text{flags}} := \{\text{out}\}$.
Else $(F_{.\text{dim}} > 0)$
    $F_{.\text{flags}} := \{\}$
    For each subface $f$ of $F$, do
        Call **Evaluate**($f$, $a$); (recursive call)
        $F_{.\text{flags}} := F_{.\text{flags}} \cup f_{.\text{flags}}$
    End
```

Figure 14: Procedure for computing **Evaluate**(F, a)

```
**ConstrainF**(F, a) Modifies $F$, returns facet $f_{new} = F \bigcap H^0$
Invariant: $F_{.\text{flags}}$ is either {in,out}, {in,on,out}, or {in,on}.

If $f_{new}$ =**ConstrainF**($F$,$a$) has already been computed Then return $f_{new}$
If $F_{.\text{dim}} = 1$ Then ($F$ is an edge with endpoints $f_1$ and $f_2$)
    Let $F ==$ subfaces$\{f_1, f_2\}$ s.t. $f_{1.\text{flags}}$ ={in} and $f_{2.\text{flags}}$ ={out} or {on}
    If $F_{.\text{flags}}$ = {in,on} Then Return $f_2$
    Else ($F_{.\text{flags}}$ = {in,out})
        Allocate a new face $f_{new}$ (of dimension 0)
        $f_{new.\text{ray}}$ =**Combine**($a, f_{1.\text{ray}}, f_{2.\text{ray}}$)
        Remove $f_2$ from subfaces of $F$
        Add $f_{new}$ to subfaces of $F$
Else ($F_{.\text{dim}} > 1$)
    If $F_{.\text{flags}}$ ={in,out}, or {in,on,out} Then
        Allocate a new face $f_{new}$
        For each facet $f$ of face $F$ Do
            If $f_{.\text{flags}}$={in} Then (do nothing– $f$ continues to be a subface)
            Else if $f_{.\text{flags}}$={out} or {on,out} Then Remove $f$ from subfaces of $F$
            Else ( $f_{.\text{flags}}$ ={in,on},{in,out},or {in,on,out})
                $g$ = **ConstrainF**($f$, $a$) (recursive call)
                If $g \neq$ null Then Add $g$ to subfaces of $f_{new}$
        End
        Add $f_{new}$ to subfaces of $F$
    Else ( $F_{.\text{flags}}$ ={in,on})
        For each facet $f$ of face $F$ Do
            If $f_{.\text{flags}}$ ={in} or {in,on} Then (do nothing– $f$ continues to be a subface)
            Else if $f_{.\text{flags}}$ ={on} Then $f_{new} = f$ ($f$ continues to be a subface)
        End
Return $f_{new}$
```

Figure 15: Procedure for computing **ConstrainF**(F, a)

7 Example

This algorithm has been implemented in C and an example is shown here. The lattice shown in figure 2 was generated from the system of constraints
$\{\ i,j,k \mid 0 \leq i \leq 1;\ 0 \leq j \leq 1;\ 0 \leq k \leq 1\ \}$. The following is a printed representation of the data structure which was created by the algorithm.

```
face(3) ed30
|
+-- face(2) ebb0
|   |
|   +-- face(1) eb10
|   |   |
|   |   +-- face(0) ca90 = point[0] [0,0,0]
|   |   |
|   |   +-- face(0) eeb0 = point[4] [1,0,0]
|   |
|   +-- face(1) eb70
|   |   |
|   |   +-- face(0) ca90 = point[0] [0,0,0] (LINK)
|   |   |
|   |   +-- face(0) ebf0 = point[1] [0,1,0]
|   |
|   +-- face(1) ee90
|   |   |
|   |   +-- face(0) eeb0 = point[4] [1,0,0] (LINK)
|   |   |
|   |   +-- face(0) ee10 = point[5] [1,1,0]
|   |
|   +-- face(1) ed00
|       |
|       +-- face(0) ebf0 = point[1] [0,1,0] (LINK)
|       |
|       +-- face(0) ee10 = point[5] [1,1,0] (LINK)
|
+-- face(2) eca0
|   |
|   +-- face(1) eb10 (LINK)
|   |
|   +-- face(1) ec60
|   |   |
|   |   +-- face(0) ca90 = point[0] [0,0,0] (LINK)
|   |   |
|   |   +-- face(0) f010 = point[2] [0,0,1]
|   |
|   +-- face(1) ef00
|   |   |
|   |   +-- face(0) eeb0 = point[4] [1,0,0] (LINK)
|   |   |
|   |   +-- face(0) f040 = point[6] [1,0,1]
|   |
|   +-- face(1) eff0
|       |
|       +-- face(0) f010 = point[2] [0,0,1] (LINK)
|       |
|       +-- face(0) f040 = point[6] [1,0,1] (LINK)
|
+-- face(2) ed70
|   |
|   +-- face(1) eb70 (LINK)
|   |
|   +-- face(1) ec60 (LINK)
|   |
|   +-- face(1) ef60
|   |   |
|   |   +-- face(0) ebf0 = point[1] [0,1,0] (LINK)
|   |   |
|   |   +-- face(0) f0c0 = point[7] [0,1,1]
|   |
|   +-- face(1) f090
|       |
|       +-- face(0) f010 = point[2] [0,0,1] (LINK)
|       |
|       +-- face(0) f0c0 = point[7] [0,1,1] (LINK)
|
+-- face(2) ee70
|   |
|   +-- face(1) ee90 (LINK)
|   |
|   +-- face(1) ef00 (LINK)
|   |
|   +-- face(1) efb0
|   |   |
|   |   +-- face(0) ee10 = point[5] [1,1,0] (LINK)
|   |   |
|   |   +-- face(0) f140 = point[8] [1,1,1]
|   |
|   +-- face(1) f110
|       |
|       +-- face(0) f040 = point[6] [1,0,1] (LINK)
|       |
|       +-- face(0) f140 = point[8] [1,1,1] (LINK)
|
+-- face(2)ece0
|   |
|   +-- face(1) ed00 (LINK)
|   |
|   +-- face(1) ef60 (LINK)
|   |
|   +-- face(1) efb0 (LINK)
|   |
|   +-- face(1) f180
|       |
|       +-- face(0) f0c0 = point[7] [0,1,1] (LINK)
|       |
|       +-- face(0) f140 = point[8] [1,1,1] (LINK)
|
+-- face(2) edc0
    |
    +-- face(1) eff0 (LINK)
    |
    +-- face(1) f090 (LINK)
    |
    +-- face(1) f110 (LINK)
    |
    +-- face(1) f180 (LINK)
```

8 Summary

The inductive method for constructing the face lattice which has been presented in this paper differs from the Seidel method in its inductive approach but shares the same order of execution time. The inductive approach is a natural consequence of the recursive structure of the face lattice. This method can be viewed as an extension of the Motzkin algorithm to compute the dual of a polyhedron. The procedure starts with a mixed system of constraints and produces an interlinked data structure representing the lattice. Execution time for this algorithm is lower bounded by the size of the output, which is $\mathcal{O}(k^{\lfloor \frac{d}{2} \rfloor})$ where k is the number of inequalities and d is the dimension.

References

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Volume 10 of *Monographs on Theoretical Computer Science*, Springer-Verlag, Berlin, 1987.

[Gol56] A. J. Goldman. Resolution and separation theorems for polyhedral convex sets. In H. W. Kuhn and A. W. Tucker, editors, *Linear inequalities and related systems*, Princeton University, Princeton,NJ, 1956.

[Gru67] B. Grunbaum. *Convex Polytopes.* Volume 16 of *Pure and Applied Mathematics*, John Wiley & Sons, London, 1967.

[McM70] P. McMullen. The maximum number of faces of a convex polytope. *Mathematica*, XVII:179–184, 1970.

[MR80] T. H. Mattheiss and D. Rubin. A survey and comparison of methods for finding all vertices of convex polyhedral sets. *Mathematics of Operations Research*, 5(2):167–185, May 1980.

[MRTT53] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. The double description method. *Theodore S. Motzkin: Selected Papers*, 1953.

[MS71] P. McMullen and G. C. Shepard. Convex polytopes and the upper bound conjecture. In *London Mathematical Society Lecture Notes Series*, Cambridge University Press, London, 1971.

[Sch86] A. Schrijver. *Theory of linear and integer programming.* John Wiley and Sons, NY, 1986.

[Sei91] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6:423–434, 1991.

[Wil93] D. Wilde. *A library for Doing Polyhedral Operations.* Master's thesis, Oregon State University, Corvallis, Oregon, Dec 1993.

Éditeur

INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)

ISSN 0249-6399