A note on Chernikova's Algorithm

H. Le Verge*

July 27, 1994

Abstract

This paper describes an implementation of Chernikova's algorithm for finding an irredundant set of vertices for a given polyhedron defined by a set of linear inequalities and equations. This algorithm can also be used for the dual problem: given a set of extremal rays and vertices, find the associated irredundant set of facet supporting hyperplanes. The method is an extension of initial Chernikova's algorithm (nonnegative domain), and is mainly based on the polyhedral cone duality principle. A new enhancement for extremal ray detection is presented together with its effects on a class of polyedra.
KEY WORDS: System of linear inequalities and equations, polyhedral cones, duality, homogeneous coordinates, bidirectional coordinates, Chernikova's algorithm.

1 Introduction

This paper is concerned with Chernikova's algorithm [Che64] [Che65] [Che68] for finding an irredundant set of vertices and rays of a given polyhedron defined by a mixed system of linear equations and inequalities $\{x|Ax = b,\ Cx \geq d\}$. Based on cone duality, this algorithm can also be used for the converse problem: finding the facets of a polytope defined by its vertices. There exist some other algorithms which can be used for solving the same problem. For a survey of these, the reader is referred to [MR80] [Chv83]. Some of these

*This work was partially funded by the French Coordinated Research Program C^3 and by the Esprit BRA project no 3280.

algorithms are based on the simplex method, and indeed can only compute nonnegative solutions. An encoding of the polyhedra in a higher dimensional space can circumvent this problem. Chernikova's algorithm essentially works on cones, but can take as input either of the following equivalent descriptions with a very small encoding cost:

$$\{x|x \geq 0,\ Ax \geq 0\} \quad (1)$$

$$\{x|Ax \geq 0\} \quad (2)$$

$$\{x|x \geq 0,\ Ax = 0\} \quad (3)$$

where A is real valued matrix.

Section 2 presents the definition and properties of cones as well as the skeleton of the algorithm. Each of the three differents representations (1) (2) (3) are discussed, respectively, in section 3, 4 and 5. Moreover, a new enhancement of the algorithm is also proposed in these sections. The encoding of any polyhedron as a cone in a higher dimensional space for finding the set of its vertices and extremal rays as well as the dual problem is presented in section 6. Section 7 shows how to find the set of general solutions of a linear programming problem. Finally, the performance of the algorithm is discussed in section 8. Appendix A contains the full source code of an implementation of this algorithm.

2 Principle of Chernikova's algorithm

Any nonempty convex polyhedral cone C can either be represented by a constraint system $\{x|Ax \geq 0\}$ or by the finite set $\{y_1, \ldots y_r\}$ of its extremal rays. Any element x belonging to C can be written:

$$x = \lambda_1 y_1 + \ldots + \lambda_r y_r$$

with $\lambda_i \geq 0$ for $1 \leq i \leq r$. The *lineality space* of a cone C, denoted $lin.space\ C$ is defined by its set of *bidirectional rays* $\{x|Ax = 0\}$. If this set equals $\{0\}$ then the cone is said to be *pointed*. For a non pointed cone, another parametric definition is given by:

$$C = Q + lin.space\ C$$

where Q a is pointed cone. It follows that:

$$x = \lambda_1 y_1 + \ldots + \lambda_r y_r + \mu_1 z_1 + \ldots + \mu_t z_t \quad \forall x \in C$$

where $y_i \in Q$, $z_j \in lin.space\ C$ and $\lambda_i \geq 0$ for $1 \leq i \leq r$. The vectors y_i are also called *unidirectional rays* as only nonnegative combinations of them belong to C.

Let A be an $m \times n$ matrix, a_k be the kth row of A and H_k the halfspace defined by $\{x | a_k x \geq 0\}$. The Chernikova's algorithm works in an incremental manner by defining recursively C as follows:

$$\begin{aligned} C_0 &= R^n \\ C_k &= C_{k-1} \cap H_k \quad \forall k\ 1 \leq k \leq m \end{aligned}$$

At each step the associated set of extremal rays Q_k is computed, and the final result is obtained for $C = C_m$. The following sections will discuss the way the set Q_k is computed depending on the formulation of the set of constraints.

Here are some ingredients which will help the reader to understand the basic idea of the algorithm. For more details see [Sch86] [NW88].

In the following a constraint $ax \geq 0$ is said to be *saturated* by a ray y if $ay = 0$. The ray y *verifies* the constraint a, if $ay \geq 0$, otherwise y *violates* the inequality. For any set X, $|X|$ denotes the cardinality of X. If Q is a set of vectors, *cone* Q denotes the set of all nonnegative combinations of vectors of Q.

Let $C = \{x | Ax \geq 0\}$ be a polyhedral cone and let y be a vector in C, $S(y, A)$ denotes the set of constraints saturated by y. In other words:

$$S(y, A) = \{a | a \text{ row of } A,\ ay = 0\}$$

Proposition 2.1 *Let y_1 and y_2 be two vectors of C, and let y be a positive combination of y_1 and y_2. Then $S(y, A) = S(y_1, A) \cap S(y_2, A)$*

Proof: As $y = \lambda y_1 + \mu y_2$ with $\lambda > 0$ and $\mu > 0$, $ay = 0$ is equivalent to $ay_1 = 0$ and $ay_2 = 0$. Therefore $\forall a$ row of A, $a \in S(y, A) \iff a \in S(y_1, A) \cap S(y_2, A)$. ■

Let $H = \{x | cx = 0\}$ be a hyperplane such that $cy_1 > 0$ and $cy_2 < 0$. Then, it is always possible to define y as a positive combination of y_1 and y_2 such that $cy = 0$. One can take $\lambda = -cy_2$ and $\mu = cy_1$.

A vector y in a set Q is called *redundant* with respect to this set if y is a nonnegative combination of the remaining vectors of Q. An *irredundant set* Q is a set with no redundant vectors. The removal of a ray from a set Q can make other redundant rays irredundant.

A *face* F of a cone C is the subset of all vectors y_i in C such that $cy_i = max\{cx|x \in C\}$ for a given vector c, provided this maximum is finite. Each face F can be represented as $\{x \in C|cx = 0\}$. It is well known that the set of all faces of a cone is a lattice under inclusion. Two faces of dimension d are *adjacent* if they are contained in a face of dimension $d+1$.

The only *minimal face* (relative to inclusion) of a cone is its lineality space. If the set $G = cone\{y\} + lin.space\ C$ is a face of a cone C then y is called an *extremal ray* of C, and G is a *minimal proper face* of C. In this case one says that y *generates* G. It can be shown that any minimal proper face of a cone $C = \{x|Ax \geq 0\}$ in n-dimensional space can also be represented as:

$$G = \{x|A'x = 0, a_i x \geq 0\} \tag{4}$$

where A' is a submatrix of A, a_i a row of A, and the rank of $\begin{bmatrix} A' \\ a_i \end{bmatrix}$ is equal to $n-t$. Here t denotes the dimension of the lineality space of C. Moreover, $lin.space\ C = \{x|a_i x = 0,\ A'x = 0\}$.

Proposition 2.2 *Two extremal rays y' and y'' in a cone $C = \{x|Ax \geq 0\}$ generate the same minimal proper face of C if and only if $S(y', A) = S(y'', A)$.*

Proof: Let $G' = cone\{y'\} + lin.space\ C$ and $G'' = cone\{y''\} + lin.space\ C$. Any vector y in G' can be written as $y = \lambda y' + \mu_1 z_1 + \ldots + \mu_t z_t$ with $z_j \in lin.space\ C$ and $\lambda \geq 0$. As $az_j = 0$ for all j by definition of the lineality space it follows that for all vectors in G', $S(y, A) = S(y', A)$. Therefore $G' = G''$ implies $S(y', A) = S(y'', A)$.

Conversely, suppose the two faces G' and G'' are distinct. Without loss of generality y'' does not belong to the set $G' = \{x|A'x = 0, a_i x \geq 0\}$. As $y'' \in \{x|a_i x \geq 0\}$ it follows that there is a constraint $a_j x = 0$ from $A'x = 0$ violated by y''. Hence $S(y', A) \neq S(y'', A)$ and the proposition is proved. ■

In fact, this proposition can be extended for more general faces, but only this case is of importance for the following.

Proposition 2.3 *Let $C = \{x|Ax \geq 0\}$ be a nonempty polyhedral cone. Then a ray $y \in C \backslash lin.space\ C$ is an extremal ray of C if and only if there is no ray y' in $C \backslash lin.space\ C$ such that $S(y, A) \subset S(y', A)$.*

Proof: Let $Q = \{y_1, \ldots, y_r\}$ be an irredundant set of extremal rays. Let $y = \lambda_1 y_1 + \ldots + \lambda_r y_r + \mu_1 z_1 + \ldots + \mu_t z_t$, where $\lambda_i > 0$ and $z_j \in lin.space\ C$. As y does not belong to $lin.space\ C$ there is at least one λ_i which is not equal to 0. If there is only one, say λ_l, then the minimal proper face generated by y_l is the same as the one generated by y. Therefore y is extremal and $S(y, A) = S(y_l, A)$ by proposition 2.2. Suppose now that there are at least two λ_i which are non zero. Then by proposition 2.1 :

$$S(y, A) \;=\; \bigcap_{i, \lambda_i \neq 0} S(y_i, A)$$

As all minimal proper faces are different, all $S(y_i, A)$ are also different. It follows that $S(y, A) \subset S(y_i, A)$ for all these i, and y is not extremal. ■

Proposition 2.4 *Let Q be a subset of $\{x|Ax \geq 0\}$ containing all extremal rays but those of the lineality space. Then a vector y is irredundant in Q if and only if there is no $y' \in Q$ such that $S(y, A) \subseteq S(y', A)$.*

Proof: This proposition follows directly from propositions 2.2 and 2.3. If y is an extremal ray then y is redundant if and only if there is another y' such that $S(y, A) = S(y', A)$. If y is not extremal then there exist at least two extremal rays y_1 and y_2 such that $S(y, A) \subset S(y_1, A)$ and $S(y, A) \subset S(y_2, A)$. ■

Given a cone C defined by its set of constraints Chernikova's algorithm seeks an irredundant set of extremal rays or associated minimal proper faces. As a consequence it will be assumed in the following, without loss of generality, that for two extremal rays y_1 and y_2, $S(y_1, A) = S(y_2, A)$ if and only if $y_1 = y_2$, as they generates the same minimal proper face.

3 Finding non-negative solutions

This section presents in more detail how the algorithm works for the case of a pointed cone brought to the form $\{x|x \geq 0,\ Ax \geq 0\}$. The different strategies used by several authors are also discussed, and finally an enhancement of the method is presented in subsection 3.1.

If Q denotes an irredundant set of unidirectional rays at step k then Q' denotes the respective set at step $k+1$. In this case (non-negative solutions) R^n_+ is taken as the initial C_0, and the associated Q_0 is equal to $\{e_1, \ldots, e_n\}$, the set of the canonical basis vectors.

Let $C = \{x | x \geq 0,\ Ax \geq 0\}$ be a pointed polyhedral cone, and $H = \{x | cx \geq 0\}$ be a halfspace of R^n. Let $Q = \{y_1, \ldots, y_r\}$ be the irredundant set of extremal rays of C, i.e. $C = cone\{y_1, \ldots, y_r\}$. Three sets can be defined with respect to the product cy_i:

$$\begin{aligned} Q^= &= \{y | y \in Q,\ cy = 0\} \\ Q^> &= \{y | y \in Q,\ cy > 0\} \\ Q^< &= \{y | y \in Q,\ cy < 0\}. \end{aligned}$$

The irredundant set Q' of extremal rays of the new cone $C \cap H$ is then equal to:

$$Q' = Q^= \cup Q^> \cup \overline{Q}$$

where $\overline{Q}$ is defined by:

$$\{y | cy = 0,\ y = \lambda y_1 + \mu y_2,\ (y_1, y_2) \in Q^> \times Q^<, adjacent(y_1, y_2),\ \lambda > 0\}$$

The first two sets $Q^=$ and $Q^>$ are composed of the rays of Q which belong to $C \cap H$. The last set $\overline{Q}$ corresponds to positive combinations of adjacent extremal rays belonging to the hyperplane $cx = 0$. The adjacency property ensures that all new rays are extremal. In [BG81] adjacency is associated with *join meet uniqueness* over the face lattice of the polyhedron. In fact new rays belongs to the hyperplane $\{x | cx = 0\}$ and also to $(t+2)$-dimensional faces of C, where t is the dimension of the lineality space.

If the adjacency property is not taken into account, it can be shown that the resulting set Q' contains all extremal rays. Of course this set is not irredundant in general, and the various implementations of Chernikova's algorithm differ in the way the redundant rays are removed (or detected). The way the set $\overline{Q}$ is constructed is a major factor for the efficiency of the algorithm.

Some earlier algorithms remove redundant rays at the end of the whole algorithm, which is by no means realistic due to combinatorial explosion. We describe here some variation on the computation of Q'. In the following the matrix D is the matrix associated with the full system of constraints defining the cone C (i.e $D = \begin{bmatrix} I \\ A \end{bmatrix}$ where I is the identity matrix.)

- The algorithm described in [Che65] use the following rule: *If there are only two rays in Q with $|Q^>| = |Q^<| = 1$ and $|Q^=| = 0$, combine them. Two rays y_1 and y_2 are adjacent if $|S(y_1, D) \cap S(y_2, D)| \geq 1$ and there is no distinct y' in Q such that $S(y_1, D) \cap S(y_2, D) \subseteq S(y', D)$.*

  The test $|S(y_1, D) \cap S(y_2, D)| \geq 1$ is a criterion which ensures that the new ray belongs to the boundary of the convex cone. If this is not the case, the ray is not added to $\overline{Q}$. Otherwise one must enumerate all Q to check the second property.

- The algorithm presented in [Hal79] checks whether there exists a ray $y' \in Q^> \cup Q^= \cup \overline{Q}$ such that $S(y_1, D) \cap S(y_2, D) \subseteq S(y', D)$. In this particular case, some rays are added incrementally to $\overline{Q}$, while some are removed because afterwards they are detected as redundant.

- In [FQ88] the set $\overline{Q}$ is generated disregarding the adjacency property. This set is merged to $Q^= \cup Q^>$, which is then sorted according to the number of saturated constraints. Finally a global redundancy check is done on this new set at each step. This particular algorithm incorporates also a heuristical constraint selection rule: the most restrictive constraint is choosen first. In fact, the order the constraints are selected may influence the intermediate number of rays, the computational speed and the necessary memory size.

At each step of the algorithm (for each constraint) and for each couple of $Q^> \times Q^<$ one must enumerate over the set Q which generally grows exponentially. In the version of Chernikova [Che65], this enumeration is avoided in one case: the number of common saturated constraints by the two combined rays is 0.

3.1 New criterion to avoid enumeration

The previous criterion can be enhanced, because the lower bound of saturated constraints of any extremal ray is generally greater than 1.

Proposition 3.1 *Let $C = \{x|Ax \geq 0\}$ be a nonempty pointed polyhedral cone, and $H = \{x|cx \geq 0\}$ be a hyperplane in an n-dimensional space such that $C \cap H \neq \emptyset$. Let y be a ray resulting from a positive combination of two extremal rays y_1 and y_2 of C such that $cy = 0$. If $|S(y, A)| \leq n - 3$ then y is not an extremal ray of $C \cap H$.*

Proof: According to the definition of a minimal proper face (4), any extremal ray y' of C saturates at least $n - t - 1$ constraints in $Ax \geq 0$, where t is the dimension of the lineality space. In this particular case $t = 0$ and $|S(y', B)| \geq n - 1$, where B is the matrix $\begin{bmatrix} A \\ c \end{bmatrix}$. As $cy = 0$, it follows that if y is an extremal ray of $C \cap H$, $|S(y, A)| \geq n - 2$. ■

The enumeration process is only necessary for rays y which saturate at least $n - 2$ constraints in C, otherwise the considered ray is immediately rejected. The number of rays thus rejected is in fact very large compared to the previous criterion.

4 Finding general solutions

Algorithms based on the simplex methods only consider nonnegative pointed cones, and polyhedra needs to be coded in a higher dimensional space to fulfill this condition by addition of slack variables. Chernikova's algorithm can find nonnegative solutions as well as negative ones and the cost of the encoding is very small compared to other algorithms. In fact, one needs only to distinguish bidirectional rays from unidirectional ones. This extension has already been proposed in [FQ88].

Let $C = \{x|Ax \geq 0\}$ be a polyhedral cone and $H = \{x|cx \geq 0\}$ be a halfspace of R^n. As the cone is not necessarily pointed (as was the case in section 3), the lineality space does not reduce to $\{0\}$. So let $Q = \{y_1, \ldots, y_r\}$ be the set of unidirectional rays, and $E = \{z_1, \ldots, z_t\}$ be the set of bidirectional rays of C. Moreover let Q' and E' denote the same sets associated with $C \cap H$.

Note that $E_0 = \{e_1, \ldots, e_n\}$ the set of canonical basis vectors, and $Q_0 = \emptyset$.
At each step, two cases may arise:

case A: There exists z_k such that $cz_k \neq 0$; then

$$\begin{aligned} Q' &= \{y'_1, \ldots, y'_r, z'_k\} \\ E' &= \{z'_1, \ldots, z'_{k-1}, z'_{k+1}, \ldots, z'_t\} \end{aligned}$$

where z'_i, is defined by:

$$\begin{aligned} z'_k &= \pm z_k \\ cz'_k &> 0 \\ z'_i &= \lambda z_i + \mu z_k, \ \forall i \neq k \\ cz'_i &= 0, \ \forall i \neq k \end{aligned}$$

and y'_j is defined by:

$$\begin{aligned} cy'_j &= 0 \\ y'_j &= \lambda y_j + \mu z_k, \ \lambda > 0. \end{aligned}$$

The vectors z_i $(i \neq k)$ and y_j are the projections of vectors z_i and y_j, respectively, on the hyperplane $cx = 0$ along the vector z_k, which leads to an equivalent cone C. This comes from the fact that extremal rays are stable when combined with any vector of the lineality space. Now, the only vector which does not belong to $C \cap H$ is z_k. Thus z_k is removed from the set of bidirectional rays, and added to the set of unidirectional rays.

Case B: All bidirectional rays are such that $cz_j = 0$; the set E of bidirectional rays is unchanged, and the transformation of Q is the same as in the case of non-negative solutions. However, the criterion defined in section 3 is changed as follows: *The enumeration can be avoided if the set of common saturated constraints contains at most $n - t - 3$ constraints, where t is the number of bidirectional rays. However, the enumeration is done in a similar way.*

5 Mixed equations and inequalities

The problem of finding solutions of a cone brought to the form $\{x|x \geq 0,\ Ax = 0\}$ has already been studied in [Che64]. Here the case of a system of mixed equations and inequalities $\{x|Ax = 0, Bx \geq 0\}$ is presented (see also [FQ88]). Although any equation $ax = 0$ can be coded by two inequalities $ax \geq 0, -ax \geq 0$, the treatment relative to an equation is generally faster than the one relative to two inequalities. As inequalities are treated in a similar way, only the case of equations is discussed.

Let $C = \{x|Ax \geq 0\}$ be a polyhedral cone, $H = \{x|cx = 0\}$ be a hyperplane of R^n, Q and E be defined as in the previous section. As usual we consider two cases:

case A': if there exists a z_k such that $cz_k \neq 0$, then the process defined in section 4 (Case A) is applied except that the positive part of z_k is not added to the set of unidirectional rays. In this case, all projected rays saturate the new constraint.

case B': If all z_k are such that $cz_k = 0$, the set E does not change and Q' is equal to $Q^= \cup \overline{Q}$.

6 Encoding polyhedra

The computation of an irredundant set of extremal rays given the constraint system associated with a cone has been presented in the previous sections. Now the encoding of any polyhedron as a cone is described for the primal problem (finding vertices) as well as for the dual problem (finding facets).

Any nonempty polyhedron $P = \{x|Ax \geq b\}$ where A is an $m \times n$ matrix and b a column m vector can be represented as

$$P \;=\; conv.hull\{x_1, \ldots, x_v\} + cone\{y_1, \ldots, y_r\} + lin.space\ P$$

where $conv.hull\ X$ denotes the set of all convex combinations of vectors of X, i.e. x belongs to $conv.hull\ X$ where $X = \{x_1, \ldots, x_v\}$ if x can be written as:

$$x \;=\; \lambda_1 x_1 + \ldots + \lambda_v x_v$$

where $1 \geq \lambda_i \geq 0$ for all i and $\sum_{i=1}^{v} \lambda_i = 1$.

The system of linear inequalities $Ax \geq b$ is transformed into an equivalent homogeneous one:

$$\left\{ \begin{array}{lcl} [A \;\; -b]xh & \geq & 0 \\ xh & \geq & 0 \end{array} \right. \tag{5}$$

where xh is a column $(n+1)$-vector. The polyhedron P is the projection over the n first coordinates of the cone defined by system (5) intersected with the set $\{xh | xh_{n+1} = 1\}$. Obviously the polyhedron is empty if all extremal rays yh of the associated cone C are such that $yh_{n+1} = 0$. If there exists at most one ray yh such that $yh_{n+1} > 0$, the lineality space of P is the projected lineality space of C. The extremal rays y of P are the projected extremal rays yh of C such that $yh_{n+1} = 0$. Finally, the vertices of P are associated with the extremal rays of the cone C such that $yh_{n+1} > 0$. More precisely, for any vertex x of P, $x_i = yh_i / yh_{n+1}$.

For the converse problem, (i.e. finding the set of constraints from extremal rays and vertices) the encoding is the following:

- for each bidirectional ray z of P define an equation $z^T a^T = 0$,
- for each unidirectional ray y of P define an inequality $y^T a^T \geq 0$,
- for each (rational) vertex x define an inequality $x'^T a^T + b \geq 0$, where $x'_i = bx_i$ with $b > 0$,

where x^T denotes the transpose of x. Solving this system, the resulting bidirectional (unidirectional) rays are associated with equations (respectively inequalities). Due to the encoding, a redundant inequality may appear in some cases. By linear combination with the equations it can be brought to the form $1 \geq 0$ (which is redundant in the polyhedron but not in the cone). A post-processing has to remove it.

7 Linear programming

As noticed by Chernikova [Che68] the same algorithm can be used for finding the set of all solutions to a linear programming problem:

$$max\{cx | Ax \geq b\}$$

Three cases may arise:

- if the associated polyhedron P is empty, the problem is of course infeasible.

- if there exists a bidirectional ray z such that $cz \neq 0$ or if there exists a unidirectional ray y such that $cy > 0$ then the maximum is unbounded.

- if all bidirectional rays z are such that $cz = 0$ and all unidirectional rays y are such that $cy \leq 0$ then the maximum is defined by $max\{cx|x$ vertex of $P\}$, and the set of solutions is

$$S = conv.hull\{x'_1, \ldots, x'_k\} + cone\{y'_1, \ldots, y'_r\} + lin.space\ P$$

  where x'_i are the vertices attaining this maximum and y'_i are such that $cy'_i = 0$.

8 Conclusion and performances

We only present an overview of the performances of the implementation given in appendix A compared with those of the original algorithm of Chernikova. Only the consequences of the the enhancement of subsection 3.1 is discussed.

The effect of the improvement is presented on hypercubes whose particular instances are given by the set of constraints $P_n = \{y \in Q^n | \alpha \geq y_i \geq 0\}$, with α a positive integer. Here y_i denotes the ith component of y. One can easily verify that on this class of polyedra, both the criteria $|S(y, A)| \geq 1|$ (initial version) and $|S(y, A)| \geq n - t - 2$ (enhanced version) are not useful as all combinations from $Q^<$ and $Q^>$ have to be generated. But if the constraint $y_1 + y_2 \geq 1$ (assuming $\alpha > 1$) is added to the polyedron P_n the time for the two algorithms differs only in the computation of this particular constraint. Considering the last step (i.e. the constraint $y_1 + y_2 \geq 1$) for the given polyedron of size n:

$$\begin{aligned} |Q^<| &= 2^{n-2} \\ |Q^>| &= 2^n - 2^{n-2} \\ |Q^=| &= 0 \end{aligned}$$

So there are $|Q^<| \times |Q^>| = 3 \times 2^{2n-4}$ combinations to examine from which only 2^{n-1} lead to an extremal ray. The new criterion $|S(y, A)| \geq n - t - 2$

rejects all combination but the 2^{n-1} necessary ones. The initial criterion $|S(y, A)| \geq 1|$ only rejects 2^{n-2}. This hence leads to $3 \times 2^{3n-4} - 2^{2n-2}$ tests of the form $S(y, A) \subseteq S(y', A)$ instead of only 2^{2n-1}.

dimension	8	9	10	11	12	13	14
IV	1.9 s	12 s	1 mn 27	12 mn	1 h 28 mn	hours	days
NV	0.5 s	2.2 s	9.3 s	39 s	2 mn 45 s	12 mn	50 mn

This table shows the computation time for the complete polyedra: the first line (IV) is relative to the initial version of Chernikova's algorithm and the second one (NV) is relative to the algorithm enhanced with the new criterion.

In fact the gain obtained is not so important in general because this classe of polyedra is very particular. Meanwhile this example highligh the great importance of finding and using good properties of polyedra (adjacency for instance). This enhancements can also be combined with the constraint selection rule presented in [FQ88], if there are many redundant inequalities. But the computation necessary for choosing the constraints is very costly. One can also add appropriate tests for particular classes of polyedra. The implementation given in appendix A incorporates this new enhancement but no constraint selection rule.

9 Acknowledgements

The author would like to thank Reinhardt Euler and Patrice Quinton for their helpful comments and suggestions.

References

[BG81] Achim Bachem and Martin Grötschel. Characterization of adjacency of faces of polyedra. *Mathematical programming study*, 14:1–22, 1981.

[Che64] N.V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of a system of linear equations. *U.S.S.R Computational Mathematics and Mathematical Physics*, 4(4):151–158, 1964.

[Che65] N.V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of a system of linear inequalities. *U.S.S.R Computational Mathematics and Mathematical Physics*, 5(2):228–233, 1965.

[Che68] N.V. Chernikova. Algorithm for discovering the set of all the solutions of a linear programming problem. *U.S.S.R Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.

[Chv83] Vasek Chvatal. *Linear programming. A Series of Books in the Mathematical Sciences*, W.H. Freeman and Company, New York/San Francisco, 1983.

[FQ88] F. Fernández and P. Quinton. *Extension of Chernikova's Algorithm for Solving General Mixed Linear Programming Problems.* Technical Report, IRISA - Rennes (France), 1988.

[Hal79] N. Halbwachs. Determination automatique de relations lineaires verifiees par les variables d'un programme. Thèse de 3ieme Cycle, Mars 1979.

[MR80] T.H. Matheiss and David S. 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.

[NW88] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization. Wiley-Interscience series in Discrete Mathematics*, John Wiley and Sons, 1988.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming. Wiley-Interscience series in Discrete Mathematics*, John Wiley and Sons, 1986.

A The source code

The following is the source code of Chernikova's algorithm for finding all solutions to a system of mixed equations and inequalities. It is written in C language, and works with standard input and output. The encoding and post-processing described in section 6 are responsibility of the user. For a cone C defined by the system $\{x|Ax = 0,\ A'x \geq 0\}$, where A is $m \times n$ and A' is $p \times n$ the corresponding input is:

$$\begin{array}{ll} m+p & n+1 \\ 0 & A \\ 1 & A' \end{array}$$

where 0 (1) denotes m (respectively p) column vectors. The algorithm works with rational data, but A and A' must be integral matrices. Without loss of generality one can use this algorithm on any polyedron in Q^n, as equation and inequalities can be scaled. The rays and rational vertices obtained as results are coded as described in section 6.

Example: Suppose one wants to determine the generating system of the polyedron defined by the system:

$$\begin{array}{rcrcrcr} -2x_1 & + & 2x_2 & - & x_3 & \geq & 0 \\ 4x_1 & + & 2x_1 & - & x_3 & \geq & 0 \\ 2x_1 & + & 6x_3 & - & 3x_3 & \geq & 20 \\ -6x_1 & - & 10x_2 & - & 5x_3 & \geq & 24 \end{array}$$

One has to type in:

```
>chernikova 20
5 5
1 -2  2 -1   0
1  4  2 -1   0
1  2  6 -3 -20
1 -6 10 -5 -24
1  0  0  0   1
```

where the argument 20 of the command is the number of available rays (i.e. the number of allocated rays). The program stops if at any moment the intermediate number of rays is greater than this value.

The output is given as:

```
bidirectional rays
1 4
   0    1    2    0
```

```
unidirectional rays
5 4
  -2    4    0    1
   1    3    0    1
   6    6    0    1
   1    1    0    0
  -1    2    0    0
```

There exists a bidirectional ray $[0,1,2]^T$, two unidirectional rays $[1,1,0]^T$ and $[-1,2,0]^T$ and three vertices $[-2,4,0]^T$, $[1,3,0]^T$ and $[6,6,0]^T$.

A.1 Source code

```
#include <malloc.h>

#define EXCH(mat, l1, l2, aux) \
  aux=mat.p[l1]; \
  mat.p[l1]=mat.p[l2]; \
  mat.p[l2]=aux;

#define NULL 0
#define TOP 2147483647
#define true 1
#define false 0


/* data structures */
typedef struct vector
{
  int size;
  int *p;
} vector;


typedef struct matrix
{
  int nbrows;
  int nbcolumns;
  int **p;
  int *p_init;
} matrix;
```

```
char *My_Alloc(nbelement, element_size)
     int nbelement, element_size;
{
  char *p;
  if ((p=malloc((unsigned)(nbelement*element_size)))==NULL) {
    printf("out of memory error\n");
    exit(1);
  }
  return p;
} /* My_Alloc */


void Alloc_Vector(vec, size)
     vector *vec;
     int size;
{
  vec->size=size;
  vec->p=(int *)My_Alloc(size, sizeof(int));
} /* Alloc_Vector */


void Alloc_Matrix(mat, nbrows, nbcolumns)
     matrix *mat;
     int nbrows, nbcolumns;
{
  int i, *p, **q;

  mat->nbrows=nbrows;
  mat->nbcolumns=nbcolumns;
  q=(int **)My_Alloc(nbrows, sizeof(int *));
  mat->p=q;
  p=(int *)My_Alloc(nbrows*nbcolumns, sizeof(int));
  mat->p_init=p;
  for (i=0;i<nbrows;i++) {
    mat->p[i]=p;
    p=p+nbcolumns;
  }
} /* Alloc_Matrix */


void Free_Vector(vec)
     vector *vec;
{
```

```
  free((char *)vec->p);
} /* Free_Vector */


void Free_Matrix(mat)
     matrix *mat;
{
  free((char *)mat->p);
  free((char *)mat->p_init);
} /* Free_Matrix */


void Read_Matrix(mat)
     matrix *mat;
{
  int nbrows, nbcolumns, i, j;

  (void) scanf("%d %d", &nbrows, &nbcolumns);
  Alloc_Matrix(mat, nbrows, nbcolumns);
  for (i=0;i<nbrows;i++)
    for (j=0;j<nbcolumns;j++)
      (void) scanf("%d", &(mat->p[i][j]));
} /* Read_Matrix */

void Write_Matrix(mat, dim)
     matrix *mat;
     int dim;
{
  int nbrows, i, j;

  (void) printf("%d %d\n", nbrows=mat->nbrows, dim);
  for (i=0;i<nbrows;i++) {
    for (j=0;j<dim;j++)
      (void) printf("%4d ", mat->p[i][j]);
    (void) printf("\n");
  }
} /* Write_Matrix */


void Gcd(a, b, g)
     int a, b, *g;
{
  int aux;
  if ((a==b) && (b==0)) *g=0;
```

```
  else {
    a=abs(a);
    b=abs(b);
    while (a) { aux=b%a; b=a; a=aux; }
    *g=b;
  }
} /* Gcd */


/* finds the minimum (absolute value) of vector vec */
void Vector_Min_Not_0(vec, min, index)
     vector *vec;
     int *min, *index;
{
  int m, size, i, ind, aux;

  size=vec->size;
  m=TOP;
  ind=-1;
  for (i=0;i<size;i++)
    if (vec->p[i]!=0)
      if (m>(aux=abs(vec->p[i]))) { ind=i; m=aux; }
  if (ind==-1) *min=1;
  else *min=m;
  *index=ind;
} /* Vector_Min_Not_0 */


/* computes the gcd of all components of vec */
Vector_Gcd(vec, size, g)
     int *vec, size, *g;
{
  vector vec_A;
  int min, index, *p, not_all_zero, i;

  Alloc_Vector(&vec_A, size);
  p=vec;
  for (i=0;i<size;i++)
    vec_A.p[i]=*p++;
  do {
    Vector_Min_Not_0(&vec_A, &min, &index);
    if (min==1) break;
    not_all_zero=false;
    for (i=0;i<size;i++)
```

```
      if (i!=index)
        not_all_zero|=(vec_A.p[i]%=min);
  } while (not_all_zero);
  *g=min;
  Free_Vector(&vec_A);
} /* Vector_Gcd */


/* normalize a vector in such a way that gcd(vec)=1 */
void Vector_Normalize(vec, size, dimension)
     int *vec;
     int size, dimension;
{
  int gcd, i, *p;

  Vector_Gcd(vec, dimension, &gcd);
  if (gcd>=2) {
    p=vec;
    for (i=0;i<size;i++)
      *p++/=gcd;
  }
} /* Vector_Normalize */


/* computes r3 as combination of r1 and r2 in such a way that r3[k]=0 */
void combine(r1, r2, r3, k, nbcolumns, dim)
     int *r1, *r2, *r3;
     int k, nbcolumns, dim;
{
  int aux, a1, a2, j;

  Gcd(r1[k], r2[k], &aux);
  a1=*(r1+k)/aux;
  a2=*(r2+k)/aux;
  for (j=0;j<nbcolumns;j++)
    r3[j]=a2*r1[j]-a1*r2[j];
  Vector_Normalize(r3, nbcolumns, dim);
} /* combine */


int chernikova(bid, uni, nbbidray, nbray, ineq)
     matrix *bid, *uni; /* matrices of bidirectional and unidirectional rays */
     int nbbidray, nbray; /* number of bidirectional and unidirectional rays */
     vector *ineq; /* vector of flags inequality/equation */
```

```
{
  vector commonzero, inequality;
  matrix ray, bidray;

  int nbcolumns, index_non_zero, nbcommonconstraints, max_rays;
  int i, j, k, l, m, *p, bound, inf_bound, equal_bound, sup_bound, redundant;
  int dimension; /* dimension of the space */

  bidray=*bid;
  ray=*uni;
  inequality=*ineq;
  nbcolumns=bidray.nbcolumns; /* width of both matrices uni and bid */
  max_rays=ray.nbrows;        /* size of matrix uni */
  dimension=nbbidray;

  Alloc_Vector(&commonzero, nbcolumns);

  for (k=dimension;k<nbcolumns;k++) {
    /* finds if exists, a bidirectional ray which does not saturate the
       current constraint */
    index_non_zero=-1;
    for (i=0;i<nbbidray;i++)
      if (bidray.p[i][k]!=0) {
        index_non_zero=i;
        break;
      }
    if (index_non_zero!=-1) {
      /* discards index_non_zero bidirectional ray */
      nbbidray--;
      if (nbbidray!=index_non_zero) {
        p=bidray.p[nbbidray];
        bidray.p[nbbidray]=bidray.p[index_non_zero];
        bidray.p[index_non_zero]= p;
      }
      /* Computes the new lineality space */
      for (i=0;i<nbbidray;i++)
        if (bidray.p[i][k]!=0)
          combine(bidray.p[i], bidray.p[nbbidray], bidray.p[i], k,
                  nbcolumns, dimension);
      /* add the positive part of index_non_zero bidirectional ray to
         the set of unidirectional rays */
      if (nbray==max_rays)
        return 1;
      if (bidray.p[nbbidray][k]<0)
```

```
    for (j=0;j<nbcolumns;j++)
      ray.p[nbray][j]=-bidray.p[nbbidray][j];
  else
    for (j=0;j<nbcolumns;j++)
      ray.p[nbray][j]=bidray.p[nbbidray][j];
  /* Computes the new pointed cone */
  for (i=0;i<nbray;i++)
    if (ray.p[i][k]!=0)
      combine(ray.p[i], ray.p[nbray], ray.p[i], k, nbcolumns, dimension);
  if (inequality.p[k])
    nbray++;
}
else {
  /* sort rays : 0 <= i < equal_bound : saturates the constraint
               : equal_bound <= i < sup_bound : verifies the constraint
               : sup_bound <= i < bound : does not verify */
  equal_bound=0;
  sup_bound=0;
  inf_bound=nbray;
  while (inf_bound>sup_bound) {
    if (ray.p[sup_bound][k]==0) {
      EXCH(ray, equal_bound, sup_bound, p);
      equal_bound++;
      sup_bound++;
    }
    else if (ray.p[sup_bound][k]<0) {
      inf_bound--;
      EXCH(ray, inf_bound, sup_bound, p);
   }
    else sup_bound++;
  }
  /* Computes only the new pointed cone */
  bound=nbray;
  for (i=equal_bound;i<sup_bound;i++)
    for(j=sup_bound;j<bound;j++) {
      /* computes the set of common saturated constraints */
      nbcommonconstraints=0;
      for (l=dimension;l<k;l++)
        if ((ray.p[i][l]==0) && (ray.p[j][l]==0)) {
          commonzero.p[nbcommonconstraints]=l;
          nbcommonconstraints++;
        }
      if (nbcommonconstraints+nbbidray>=dimension-2) {
        /* check whether a ray m saturates the same set of constraints */
```

```
            redundant=false;
            for (m=0;m<bound;m++)
              if ((m!=i) && (m!=j)) {
                for (l=0;l<nbcommonconstraints;l++)
                  if (ray.p[m][commonzero.p[l]]!=0) break;
                if (l==nbcommonconstraints) {
                  /* the combination of ray i and j will generate a
                     non extremal ray so ... */
                  redundant=true;
                  break;
                }
              }
            if (!redundant) {
              if (nbray==max_rays)
                return 1;
              /* computes the new ray */
              combine(ray.p[j], ray.p[i], ray.p[nbray], k, nbcolumns,
                      dimension);
              nbray++;
            }
          }
        }
      /* Eliminates all non extremal rays */
      if (inequality.p[k])
        j=sup_bound;
      else
        j=equal_bound;
      i=nbray;
      while ((j<bound)&&(i>bound)) {
        i--;
        EXCH(ray, i, j, p);
        j++;
      }
      if (j==bound) nbray=i;
      else nbray=j;
    }
  }
  Free_Vector(&commonzero);

  bid->nbrows=nbbidray;
  uni->nbrows=nbray;
  return 0;
} /* chernikova */
```

```
main(argc, argv)
     int argc;
     char **argv;
{
  matrix mat, ray, bidray;
  vector inequality; /* vector of flags equation/inequality */
  int nbconstraints, dimension, nbcolumns, nbcolumns2, i, j;

  Read_Matrix(&mat);

  nbconstraints=mat.nbrows;
  nbcolumns=mat.nbcolumns;

  dimension=nbcolumns-1;
  nbcolumns2=dimension+nbconstraints;

  Alloc_Matrix(&ray, atoi(argv[1]), nbcolumns2);
  Alloc_Matrix(&bidray, dimension, nbcolumns2);

  /* set the identity matrix as bidirectional rays */
  for (i=0;i<dimension;i++) {
    for (j=0;j<dimension;j++)
      bidray.p[i][j]=0;
    bidray.p[i][i]=1;
  }

  for (i=0;i<nbconstraints;i++)
    for (j=1;j<nbcolumns;j++)
      bidray.p[j-1][i+dimension]=mat.p[i][j];

  Alloc_Vector(&inequality, nbcolumns2);
  for (i=0;i<nbconstraints;i++)
    inequality.p[i+dimension]=mat.p[i][0];

  if (chernikova(&bidray, &ray, dimension, 0, &inequality)!=0) {
    printf("not enough rays available\n");
    exit(2);
  }

  Free_Vector(&inequality);

  printf("bidirectional rays\n");
  Write_Matrix(&bidray, mat.nbcolumns-1);
```

```
  printf("unidirectional rays\n");
  Write_Matrix(&ray, mat.nbcolumns-1);

  Free_Matrix(&bidray);
  Free_Matrix(&ray);

  return 0;
} /* main */
```