Double Description Method Revisited

Komei Fukuda[1] and Alain Prodon[2]

[1] Institute for Operations Research, ETHZ, CH-8092 Zürich, Switzerland
[2] Department of Mathematics, EPFL, CH-1015 Lausanne, Switzerland

Abstract. The double description method is a simple and useful algorithm for enumerating all extreme rays of a general polyhedral cone in $\mathbb{R}^d$, despite the fact that we can hardly state any interesting theorems on its time and space complexities. In this paper, we reinvestigate this method, introduce some new ideas for efficient implementations, and show some empirical results indicating its practicality in solving highly degenerate problems.

1 Introduction

A pair (A, R) of real matrices A and R is said to be a *double description pair* or simply a *DD pair* if the relationship

$$Ax \geq 0 \quad \text{if and only if} \quad x = R\,\lambda \text{ for some } \lambda \geq 0$$

holds. Clearly, for a pair (A, R) to be a DD pair, it is necessary that the column size of A is equal to the row size of R, say d. The term "double description" was introduced by Motzkin et al. [MRTT53], and it is quite natural in the sense that such a pair contains two different descriptions of the same object. Namely, the set $P(A)$ represented by A as

$$P(A) = \{x \in \mathbb{R}^d : Ax \geq 0\} \tag{1}$$

is simultaneously represented by R as

$$\{x \in \mathbb{R}^d : x = R\,\lambda \text{ for some } \lambda \geq 0\}. \tag{2}$$

A subset P of $\mathbb{R}^d$ is called *polyhedral cone* if $P = P(A)$ for some matrix A, and a matrix A is called a *representation matrix* of the polyhedral cone $P(A)$. We shall use the simpler term *cone* for polyhedral cone for the sequel since we only deal with such cones.

When a cone P is represented by (2), we say R is a *generating matrix* for P or a matrix R *generates* the polyhedron. Clearly, each column vector of a generating matrix R lies in the cone P and every vector in P is a nonnegative combination of some columns of R.

Minkowski's Theorem states that every polyhedral cone admits a generating matrix.

Theorem 1 (Minkowski's Theorem for Polyhedral Cones). *For any $m \times d$ real matrix A, there exists some $d \times n$ real matrix R such that (A, R) is a DD pair, or in other words, the cone $P(A)$ is generated by R.*

Here, the nontriviality is in that the row size n of R is a finite number. If we allow the size n to be infinite, there is a trivial generating matrix consisting of all vectors in the cone. In this respect, the essence of the theorem is the statement "every cone is finitely generated."

We have also the converse of the theorem, known as Weyl's Theorem.

Theorem 2 (Weyl's Theorem for Polyhedral Cones). *For any $d \times n$ real matrix R, there exists some $m \times d$ real matrix A such that (A, R) is a DD pair, or in other words, the set generated by R is the cone $P(A)$.*

These two theorems suggest two fundamental problems, one to construct a matrix R from a given matrix A and the converse. It is well known that these two problems are computationally equivalent, meaning they are linear-time reducible to each other. In fact, one can easily prove by using Farkas' Lemma that (A, R) is a DD pair if and only if (R^T, A^T) is a DD pair. Thus, we can treat the latter problem with a given R as the first problem with A being R^T. For this reason, we shall concentrate on the first problem to find a generating matrix R for a given A.

Clearly a more appropriate formulation of the problem is to require the minimality of R, meaning: find such a matrix R so that no proper submatrix is generating $P(A)$. Note that a minimal set of generators is unique up to positive scaling when we assume the regularity condition that the cone is *pointed*, i.e. the origin 0 is an extreme point of $P(A)$. Geometrically, the columns of a minimal generating matrix are in 1-to-1 correspondence with the extreme rays of P. Thus the problem is also known as the *extreme ray enumeration problem.* In this paper, this problem is the main concern since both theoretically and practically there is neither necessity nor importance to generate redundant information.

The extreme ray enumeration problem has been studied, directly or indirectly, by many researchers in mathematics, operations research and computational geometry etc. and considerable efforts have been devoted to finding better algorithms for the problem. Despite that, no efficient enumeration algorithm for the general problem is known. Here, we mean by an *efficient enumeration algorithm* one which runs in time polynomial in both the input size and the output size. Such efficient algorithms exist for certain special cases only.

The problem is said to be *degenerate* if there is a vector x in $P(A)$ satisfying more than d inequalities in $Ax \geq 0$ with equality, and *nondegenerate* otherwise. Avis and Fukuda [AF92] showed that when the problem is nondegenerate it is possible to solve the extreme ray enumeration problems in time $O(mdv)$ and in space $O(md)$ where v is the number of extreme rays. Note that the problem treated in [AF92] is the vertex enumeration problem, which is the non-homogeneous version of our problem and there are simple transformations between them. (A C-implementation of Avis-Fukuda algorithm is publicly available [Avi93].)

In this paper, we examine a classical algorithm, known as the double description (DD) method [MRTT53], which is an extremely simple algorithm. We are primarily interested in practicality of this algorithm applied to highly generate problems. Let us describe the algorithm briefly. Let A^k be the submatrix of A consisting of the first k rows of A. It is easy to find a generating matrix R^k for $P(A^k)$ when k is small, say $k = 1$. In the general iteration k, the algorithm constructs a DD pair (A^{k+1}, R^{k+1}) from a DD pair (A^k, R^k) already computed.

This algorithm was rediscovered repeatedly and given different names. The method known as Chernikova's algorithm [Che65] is basically the same method. In the dual setting of computing A from the generating matrix R (known as the convex hull computation), the method is essentially equivalent to the beneath-and-beyond method, see [Ede87, Mul94]. On the other hand, the algorithm known as the Fourier-Motzkin elimination is more general than the DD method, but often considered as the same method partly because it can be used to solve the extreme ray enumeration problem. This algorithm is essentially different because it does not use the crucial information of double description of the associated convex polyhedra even when it is applied to the problem.

The importance of the DD method is not well understood theoretically at least for the moment, since the worst-case behavior of this algorithm is known to be exponential in the sizes of input and output. The algorithm is sensitive to the permutations of rows of A, and an exponential behavior is exhibited when the (column) size of R^k explodes with some bad ordering of rows for certain class of matrices A, see [Dye83]. Yet, it has been observed that once it is properly implemented with a certain heuristic strategy of selecting an ordering and further techniques, it becomes a very competitive algorithm, in particular for highly degenerate problems. To present some useful implementation techniques for the DD algorithm is the main purpose of this paper.

In order to argue for the necessity of various implementation techniques, let us list serious pitfalls of a naive DD method which must be avoided whenever possible:

(a) the algorithm is terribly sensitive to the ordering of rows of A in practice as well and a random ordering often leads to prohibitive growths of intermediate generator matrices R^k (especially for highly degenerate problems);

(b) it generates enormous number of redundant generators and goes beyond any tractable limit of computation very quickly, even when some good ordering (a) is known;

(c) if the input data A is perturbed to resolve degeneracy, the size of perturbed output can grow exponentially in the sizes of original input and output;

(d) data structures and computational methods should be part of the algorithm, since they can significantly improve the efficiency of computation when appropriately chosen.

It should be noted the original paper [MRTT53] gives a satisfactory remedy to (b) and explains how to generate the minimum generating matrix R^k. Our implementations in fact follow their ideas. Also the point (c) has been discussed in a recent paper [AB95] where it is shown that the perturbation makes the DD method exponentially slower for a certain class of polyhedra. While this serves as the formal justification of not using perturbation techniques, we find that the DD method can exploit a very little from nondegeneracy assumptions. Thus our implementations do not apply any perturbations to input data. Our main contribution is thus to show some practical ideas to handle the difficulties (a) and (d).

With respect to (a), we tested several different orderings of rows. Orderings can be divided into two classes, the static ordering and the dynamic ordering. The former is an ordering prefixed at the beginning of the computation procedure and the latter selects the next row dynamically as the computation proceeds. A typical static ordering is the lexicographic ordering (lex-min) and a typical dynamic ordering is the "max-cut-off" ordering which amounts to select the row that removes as many generators (i.e. columns of R^k) as possible. Experimentally we found that two dynamic orderings, max-cut-off and min-cut-off, perform much worse than the lex-min ordering for relatively large set of degenerate test problems. Also, we study a static ordering which gradually increases the dimension of the intermediate cones. This ordering motivates us to introduce a decomposition technique for the DD method, called the column decomposition. This technique is especially useful for the cones arising from combinatorial optimization problems.

Concerning (d), we are proposing two different data structures. Both data structures are for efficient handling of the minimum generating matrix R consisting of extreme rays and of the adjacency among the extreme rays. Our implementations differ considerably from the data structure suggested by the books (e.g.) for the dual version (the beneath-and-beyond method) of the algorithm which stores the complete adjacency of extreme rays. The main idea in our implementations is to store only necessary adjacency relations. This makes the implementation less expensive in both storage and time. We report some experimental results showing the practicality of our implementations.

Since the DD method must store a DD pair for each intermediate cone during the computation, one can easily exhaust any memory space available in computers. For this reason, it is useful if we can decompose the given problem to smaller problems. We introduce the row decomposition technique for this purpose. This technique is useful not only for reducing the size of a given problem but for parallelizing the computation. We have not yet tested the practicality of this technique and this will be a subject of future research.

Finally we would like to emphasize that although we found our implementations quite competitive, it is not our interest to compare our implementations with the other existing implementations, e.g. [BDH93, Ver92, Wil93]. It is very difficult to make fair comparisons and we do not think such comparisons are very useful. Instead, we concentrate on showing what we learned from our experiences with the DD method, and present some techniques we found useful. One can reproduce most of our experimental findings since one of the implementations we use is publicly available by anonymous ftp [Fuk93].

2 Double Description Method: Primitive Form

In this section, we give the simplest form of the DD method [MRTT53]. Suppose that an $m \times d$ matrix A is given, and let $P(A) = \{x : Ax \geq 0\}$. The DD method is an incremental algorithm to construct a $d \times n$ matrix R such that (A, R) is a DD pair.

For the sequel of this paper, we assume for simplicity that the cone $P := P(A)$ is *pointed*, i.e., the origin 0 is an extreme point of P or equivalently A has rank d. To reduce a problem to the pointed case, one can simply restrict the cone to the orthogonal subspace of the lineality space $\{x : A\, x = 0\}$ of P. We describe in Section 4.2 how to handle directly this case. We also assume that the system $Ax \geq 0$ is irredundant.

Let K be a subset of the row indices $\{1, 2, \ldots, m\}$ of A and let A_K denote the submatrix of A consisting of rows indexed by K. Suppose we already found a generating matrix R for $P(A_K)$, or equivalently (A_K, R) is a DD pair. If $A = A_K$, clearly we are done. Otherwise we select any row index i not in K and try to construct a DD pair (A_{K+i}, R') using the information of the DD pair (A_K, R).

Once this basic procedure is described, we have an algorithm to construct a generating matrix R for $P(A)$. This procedure can be easily understood geometrically by looking at the cut-section C of the cone $P(A_K)$ with some appropriate hyperplane h in R^d which intersects with every extreme ray of $P(A_K)$ at a single point. Let us assume that the cone is pointed and thus C is bounded. Having a generating matrix R means that all extreme rays (i.e. extreme points of the cut-section) of the cone are represented by columns of R. Such a cut-section (of a cone lying in $\mathbb{R}^4$) is illustrated in Fig. 1. In this example, C is the cube $abcdefgh$.

Fig. 1. Illustration of the mechanism of the DD method.

The newly introduced inequality $A_i\ x \geq 0$ partitions the space $\mathbb{R}^d$ into three parts:

$$\begin{aligned} H_i^+ &= \{x \in \mathbb{R}^d : A_i\ x > 0\} \\ H_i^0 &= \{x \in \mathbb{R}^d : A_i\ x = 0\} \\ H_i^- &= \{x \in \mathbb{R}^d : A_i\ x < 0\}. \end{aligned} \tag{3}$$

The intersection of H_i^0 with P and the new extreme points i and j in the cut-section C are shown in bold in Fig. 1.

Let J be the set of column indices of R. The rays r_j $(j \in J)$ are then partitioned into three parts accordingly:

$$\begin{aligned} J^+ &= \{j \in J : r_j \in H_i^+\} \\ J^0 &= \{j \in J : r_j \in H_i^0\} \\ J^- &= \{j \in J : r_j \in H_i^-\}. \end{aligned} \tag{4}$$

We call the rays indexed by J^+, J^0, J^- the *positive*, *zero*, *negative* rays with respect to i, respectively. To construct a matrix R' from R, we generate new $|J^+| \times |J^-|$ rays lying on the ith hyperplane H_i^0 by taking an appropriate positive combination of each positive ray r_j and each negative ray $r_{j'}$ and by discarding all negative rays.

The following lemma ensures that we have a DD pair (A_{K+i}, R'), and provides the key procedure for the most primitive version of the DD method.

Lemma 3 (Main Lemma for Double Description Method). *Let (A_K, R) be a DD pair and let i be a row index of A not in K. Then the pair (A_{K+i}, R') is a DD pair, where R' is the $d \times |J'|$ matrix with column vectors r_j $(\ j \in J'\ )$ defined by*

$$\begin{aligned} J' &= J^+ \cup J^0 \cup (J^+ \times J^-), \text{ and} \\ r_{jj'} &= (A_i\ r_j) r_{j'} - (A_i\ r_{j'}) r_j \quad \text{for each } (j,j') \in J^+ \times J^-. \end{aligned}$$

Proof. Let $P = P(A_{K+i})$ and let P' be the cone generated by the matrix R'. We must prove that $P = P'$. By the construction, we have $r_{jj'} \in P$ for all $(j,j') \in J^+ \times J^-$ and $P' \subset P$ is clear.

Let $x \in P$. We shall show that $x \in P'$ and hence $P \subset P'$. Since $x \in P$, x is a nonnegative combination of r_j's over $j \in J$, i.e., there exist $\lambda_j \geq 0$ for $j \in J$ such that

$$x = \sum_{j \in J} \lambda_j r_j. \tag{5}$$

If there is no positive λ_j with $j \in J^-$ in the expression above then $x \in P'$. Suppose there is some $k \in J^-$ with $\lambda_k > 0$. Since $x \in P$, we have $A_i\ x \geq 0$. This together with (5) implies that there is at least one $h \in J^+$ with $\lambda_h > 0$. Now by construction, $hk \in J'$ and

$$r_{hk} = (A_i\ r_h) r_k - (A_i\ r_k) r_h. \tag{6}$$

By subtracting an appropriate positive multiple of (6) from (5), we obtain an expression of x as a positive combination of some vectors r_j $(j \in J')$ with new coefficients λ_j where the number of positive λ_j's with $j \in J^+ \cup J^-$ is strictly smaller than in the first expression. As long as there is $j \in J^-$ with positive λ_j, we can apply the same transformation. Thus we must find in a finite number of steps an expression of x without using r_j such that $j \in J^-$. This proves $x \in P'$, and hence $P \subset P'$. □

It is quite simple to find a DD pair (A_K, R) when $|K| = 1$, which can serve as the initial DD pair. Another simple (and perhaps the most efficient) way to obtain an initial DD form of P is by selecting a maximal submatrix A_K of A consisting of linearly independent rows of A. The vectors r_j's are obtained by solving the system of equations:

$$A_K \, R \;=\; I,$$

where I is the identity matrix of size $|K|$, R is a matrix of unknown column vectors r_j, $j \in J$. As we have assumed $rank(A) = d$, i.e. $R = A_K^{-1}$, the pair (A_K, R) is clearly a DD pair, since $A_K x \geq 0 \leftrightarrow x = A_K^{-1}\lambda$, $\lambda \geq 0$.

Here we write the DD method in procedural form:

> **procedure** DoubleDescriptionMethod(A);
> **begin**
>   Obtain any initial DD pair (A_K, R);
>   **while** $K \neq \{1, 2, \ldots, m\}$ **do**
>   **begin**
>   Select any index i from $\{1, 2, \ldots, m\} \setminus K$;
>   Construct a DD pair (A_{K+i}, R') from (A_K, R);
>   /* by using Lemma 3 */
> $R := R'$; $K := K + i$;
>   **end**;
>   Output R;
> **end.**

The DD method given here is very primitive, and the straightforward implementation will be quite useless, because the size of J increases very fast and goes beyond any tractable limit. One reason for this is that many (perhaps, most) vectors $r_{jj'}$ the algorithm generates (defined in Lemma 3), are unnecessary. In the next section we will show how to avoid generating redundant vectors.

The DD method in dual form, for constructing the convex hull of a finite set of points, is known as the beneath-beyond method and studied extensively in the field of computational geometry. An implementation proposed in [Mul94], when interpreted as the DD method, is basically to use random inequality ordering and to store the complete adjacencies of the rays in the intermediate cones for efficient updates. One serious problem is that the analysis which yields its "optimality" depends on the assumption that the input is nondegenerate (or dually, the points are in general position). This assumption is rather nonrealistic because there is an efficient (linear-time) algorithm [AF92] for nondegenerate case and thus problems for which the DD method is superior must be degenerate. Furthermore, we show that it is not necessary to store the adjacencies of rays. In fact one of our new technique is essentially to store only those adjacencies which are needed for the computation.

3 Practical Implementations

In this section, we shall describe three practical implementations of the DD method. The first one is the simplest and might be called standard in the sense that anyone implementing the DD method is quite likely to come up with this implementation, and is in fact suggested in the original paper [MRTT53]. We shall give new implementations which are more efficient than the standard algorithm.

3.1 The standard implementation

We first introduce some definitions and properties which we will use in strengthening Lemma 3. Remember that we are assuming that P is pointed, that is $rank(A) = d$.

A vector r is said to be a *ray* of P if $r \neq 0$ and $\alpha\, r \in P$ for all positive α. We identify two rays r and r' if $r = \alpha\, r'$ for some positive number α. We shall denote this equivalence by $r \simeq r'$. For any vector x in P, we define the *zero set* or *active set* $Z(x)$ as the set of inequality indices i such that $A_i x = 0$. The word zero comes from "slack variables being zero" at the associated inequalities.

Proposition 4. *Let r be a ray of P, $\bar{F} := \{x : A_{Z(r)}x = 0\}$, $F := \bar{F} \cap P$ and $rank(A_{Z(r)}) = d - k$. Then*

(a) *$rank(A_{Z(r)\cup\{i\}}) = d - k + 1$ for all $i \notin Z(r)$;*
(b) *F contains k linearly independent rays;*
(c) *If $k \geq 2$ then r is a nonnegative combination of two distinct rays r_1 and r_2 with $rank(A_{Z(r_i)}) > d - k$, $i = 1, 2$.*

Proof. (a) If A_i is a linear combination of rows of A_K with $K \subseteq Z(r)$, then $A_i r = 0$.

(b) Clearly $\bar{F}$ contains k linearly independent vectors r, v_2, $v_3, \ldots, v_k$. Let $r_1 := r$, $r_i := r + \alpha_i v_i$, $i = 2, \ldots, k$. These vectors are linearly independent for $\alpha_i \neq 0$. When the coefficients α_i are chosen such that $\alpha_i > 0$ and $\alpha_i \leq min(-A_j r / A_j v_i$ for all j s.t. $A_j v_i < 0)$ for $v_i \notin P$, the vectors r_i, $i = 1, \ldots, k$, are k linearly independent vectors in F.

(c) Let $k \geq 2$. $\bar{F}$ contains a vector v s.t. neither v nor $-v$ is in P : since $rank(A_{Z(r)}) \leq rank(A) - 2$, there exist i and j such that $rank(A_{Z(r)\cup\{i,j\}}) = rank(A_{Z(r)}) + 2$, and thus there exist v_1 and v_2 in $\bar{F}$ with $A_i v_1 = 0$, $A_j v_1 > 0$, $A_i v_2 < 0$, $A_j v_2 = 0$. Then $v = v_1 + v_2$ satisfies $A_i v < 0$ and $A_j v > 0$. Let $r_1 = r + \alpha_1 v_1$, $r_2 = r - \alpha_2 v_2$. Then there exist positive maximal values for α_i s.t. $r_i \in F$.

For these α_i we get $Z(r_i) \supset Z(r)$, $i = 1, 2$ and $r = 1/(\alpha_1 + \alpha_2)(\alpha_2 r_1 + \alpha_1 r_2)$. Thus (c) holds. $\square$

A ray r is said to be *extreme* if it is not a nonnegative combination of two rays of P distinct from r.

Proposition 5. *Let r be a ray of P. Then*

(a) *r is an extreme ray of P if and only if the rank of the matrix $A_{Z(r)}$ is $d-1$;*
(b) *r is a nonnegative combination of extreme rays of P.*

Proof. Observe that if there exist some $\lambda_j > 0$ and $r_j \in P$ such that $r = \sum_j \lambda_j r_j$ then each of these r_j belongs to $\bar{F} := \{x : A_{Z(r)}x = 0\}$, since for $i \in Z(r)$, $A_i r = 0$ and $A_i r$ is a sum of nonnegative terms.

If $rank(A_{Z(r)}) = d - 1$, then $\bar{F} = \{kr : k \in \mathbb{R}\}$ and 0 and $-r$ are the only elements distinct of r in $\bar{F}$, hence r is an extreme ray.

If $rank(A_{Z(r)}) < d-1$, then by Proposition 4 there exist rays r_1 and r_2 such that r is a nonnegative combination of these rays, and hence r is not an extreme ray. This proves (a). Moreover, since $rank(A_{Z(r_i)}) > rank(A_{Z(r)})$ for $i = 1, 2$, repeating the last argument for r_1 and r_2 until $rank(A_{Z(r_i)}) = d - 1$, $i = 1, 2$, proves (b). $\square$

Since every extreme ray is certainly necessary to generate P we have the following corollary.

Corollary 6. *Let R be a minimal generating matrix of P. Then R is the set of extreme rays of P.*

Observe that the assumption that P be pointed is important here, because then computing the unique generating matrix is a well formulated problem. When P is not pointed, there are infinitely many minimal generating matrices.

We say that two distinct extreme rays r and r' of P are *adjacent* if the minimal face of P containing both contains no other extreme rays. This is equivalent to say that, if r'' is an extreme ray of P with $Z(r'') \supseteq Z(r) \cap Z(r')$, then either $r'' = r$ or $r'' = r'$.

Proposition 7. *Let r and r' be distinct rays of P. Then the following statements are equivalent:*

(a) *r and r' are adjacent extreme rays;*
(b) *r and r' are extreme rays and the rank of the matrix $A_{Z(r)\cap Z(r')}$ is $d-2$.*
(c) *if r'' is a ray with $Z(r'') \supset Z(r) \cap Z(r')$ then either $r'' \simeq r$ or $r'' \simeq r'$;*

Proof. Let r and r' be distinct rays of P and $\bar{F} := \{x : A_{Z(r)\cap Z(r')}x = 0\}$. Then $F := \bar{F} \cap P$ is the minimal face of P containing r and r'.

To prove the equivalence of (a) and (b) let r and r' be extreme rays of P. Since P is pointed, we have $Z(r) \neq Z(r')$.

If (b) holds, i.e. $rank(A_{Z(r)\cap Z(r')}) = d-2$, then $dim(\bar{F}) = 2$ and thus r and r' generate $\bar{F}$, that is each x in $\bar{F}$ can be written as $x = \alpha r + \alpha' r'$. Moreover, $x \in F$ implies α and α' are nonnegative, since there exists some $i \in Z(r') - Z(r)$ (respectively $Z(r) - Z(r')$), for which $A_i x \geq 0$ and $A_i x = \alpha A_i r$ (respectively $A_i x = \alpha' A_i r'$). Thus x is a nonnegative combination of r and r' and there is no other extreme ray in F. Thus (a) holds. Observe also that, since a two dimensional face contains at least two extreme rays by Proposition 4, it contains exactly two extreme rays.
If (b) does not hold, i.e. $rank(A_{Z(r)\cap Z(r')}) = d-k$ with $k \geq 3$, then by Proposition 4(b) $dim(F) = k$ and at least k extreme rays are necessary to generate F by Proposition 5(b). Thus r and r' are not adjacent.

We now use the former equivalence to prove $(a) \Leftrightarrow (c)$.
If (a) holds, then $rank(A_{Z(r)\cap Z(r')}) = d-2$. It follows from Proposition 4(a) that $Z(r'') \supset Z(r) \cap Z(r')$ imply $rank(A_{Z(r'')}) = d-1$, i.e. r'' is an extreme ray by Proposition 5. Thus, since r and r' are adjacent, (c) holds.
If (a) does not hold, that is, either r and r' are nonadjacent extreme rays or at least one is not an extreme ray. Then if $rank(A_{Z(r)\cap Z(r')}) < d-2$ there exist in F at least one extreme ray different from r and r', which can serve as r''. If $rank(A_{Z(r)\cap Z(r')}) = d-2$ then we know from (b) that r or r' is nonextreme and that F contains two extreme rays, thus at least one of these extreme rays is different from r and r' and can serve as r'', showing the invalidity of (c). □

In the proposition above, the statement (c) can be called a combinatorial characterization of the adjacency, and (b) an algebraic characterization.

Now the main lemma 3 can be strengthened for practical purposes as follows:

Lemma 8 (Strengthened Main Lemma for the DD Method). *Let (A_K, R) be a DD pair such that $rank(A_K) = d$ and let i be a row index of A not in K. Then the pair (A_{K+i}, R') is a DD pair, where R' is the $d \times |J'|$ matrix with column vectors r_j ($j \in J'$) defined by*

$$J' = J^+ \cup J^0 \cup Adj,$$
$$Adj = \{(j,j') \in J^+ \times J^- : r_j \text{ and } r_{j'} \text{ are adjacent in } P(A_K)\}, \text{ and}$$
$$r_{jj'} = (A_i\, r_j) r_{j'} - (A_i\, r_{j'}) r_j \quad \text{for each } (j,j') \in Adj.$$

Furthermore, if R is a minimal generating matrix for $P(A_K)$ then R' is a minimal generating matrix for $P(A_{K+i})$.

Proof. Let all the assumptions be satisfied. We know from Proposition 5 that each extreme ray of $P(A_K)$ must belong to R, and that only the extreme rays of $P(A_{K+i})$ are necessary in R'. Denote for abbreviation $W := Z(r_j) \cap Z(r_{j'}) \cap K$. Observe that $Z(r_{jj'}) \cap (K+i) = W \cup \{i\}$.
If r_j and $r_{j'}$ are adjacent extreme rays of $P(A_K)$, then $rank(A_W) = d-2$. Then, by Proposition 4(a), $rank(A_{Z(r_{jj'})\cap(K+i)}) = d-1$. Thus $r_{jj'}$ is an extreme ray of $P(A_{K+i})$ and must belong to R'.

Suppose r_j and $r_{j'}$ are not adjacent extreme rays. Then if $rank(A_W) < d-2$ we get $rank(A_{Z(r_{jj'})\cap(K+i)}) < d-1$ and thus $r_{jj'}$ is not necessary. If $rank(A_W) = d-2$, then we know from Proposition 7 that r_j and $r_{j'}$ cannot be both extreme rays of $P(A_K)$. But they belong to a two dimensional face containing exactly two extreme rays of $P(A_K)$ which thus belong to R: This adjacent pair will then produce a new ray equal to $r_{jj'}$, so $r_{jj'}$ is not necessary.

Hence all new rays are extreme rays of $P(A_{K+i})$ and R' is minimal if R was minimal. □

By using this lemma, we can write a straightforward variation of the DD method which produces a minimal generating set for P:

procedure DDMethodStandard(A);
begin
    Obtain any initial DD pair (A_K, R) such that R is minimal;
    while $K \neq \{1, 2, \ldots, m\}$ **do**
    begin
        Select any index i from $\{1, 2, \ldots, m\} \setminus K$;
        Construct a DD pair (A_{K+i}, R') from (A_K, R);
            /* by using Lemma 8 */
        $R := R'$; $K := K + i$;
    end;
    Output R;
end.

To implement DDMethodStandard, we must check for each pair of extreme rays r and r' of $P(A_K)$ with $A_i\, r > 0$ and $A_i\, r' < 0$ whether they are adjacent in $P(A_K)$. As we state in Proposition 7, there are two ways to check adjacency, the combinatorial and the algebraic way. We do not know any theoretical reason to conclude which method is more efficient. Our computational experiments indicate that the combinatorial method is almost always faster than the algebraic method. See Section 5.1.

The asymptotic complexity of this algorithm depends strongly on the number of extreme rays of the intermediate cones, and thus on the ordering of the rows of A. However, if we assume that this number remains in $O(v)$, v denoting the size of the output and m the number of rows of A, then the whole complexity is in $O(m^2v^3)$, the dominating part coming from the combinatorial test applied at each iteration : for each pair (there are at most $O(v^2)$) the test makes at most $O(v)$ comparisons of vectors of size m.

3.2 A new implementation with prefixed row ordering

From our computational experiences, the variation of the DD method described in the previous section is a reasonably efficient algorithm if it incorporates a good ordering of inequalities. However this method scans all pairs (r, r') of rays of $P(A_K)$ with $A_i\, r > 0$ and $A_i\, r' < 0$ in each iteration, and generates a new ray only when they are identified adjacent in $P(A_K)$. It is natural to imagine that most of these pairs are non-adjacent pairs and thus scanning all the pairs would be very inefficient.

One way to reduce the unnecessary scanning might be to store all the adjacencies of the rays, and to scan only the adjacent pairs. This modification might look efficient but it has several serious drawbacks. The first one is that the storage space can be quite large since the number of adjacencies can be quadratic in the number of rays. Secondly, it is not clear how to select only those pairs (r, r') of rays with $A_i\, r > 0$ and $A_i\, r' < 0$ without scanning through the large adjacency data. Furthermore updating the adjacency data is quite expensive.

These observations lead us to seek for an implementation of the DD method which stores only those adjacent pairs of rays in $P(A_K)$ that produce the new rays at later iterations. We shall show that this can be done if the ordering of inequalities is prefixed.

The idea is quite simple. Suppose we fix the ordering of inequalities as the natural ordering. For convenience, let

$$P^j = \{x : A_i\, x \geq 0 \text{ for } i = 1, 2, \ldots, j\},$$

for $j = 1, 2, \ldots, m$. Thus $P^m = P$. Suppose we are at ith iteration to add the ith inequality and the remaining inequality indices are $i+1, i+2, \ldots, m$. At this iteration, a minimal generating set R for the cone P^{i-1} is at hand and the DD method will produce a new set of rays satisfying the ith inequality with equality and produce a minimal generating set for the cone P^i. Let (r, r') be a pair of adjacent rays in P^i. Since the ordering of inequalities is fixed, we know exactly what will happen to this pair. Namely, there are three cases.

(a) Both r and r' are rays of P (and thus this adjacency won't create any new rays);

(b) Both r and r' stay rays of P^j for $j = i, i+1, \ldots, k-1$, and both becomes infeasible for P^k for some $k \leq m$ (and thus this adjacency won't create any new rays either);

(c) Both r and r' stay rays of P^j for $j = i, i+1 \ldots, k-1$, and only one of the rays, say r, becomes infeasible for P^k for some $k \leq m$. (A new ray is created only when the other ray r' satisfies the kth inequality with strict inequality, i.e. $k \notin Z(r')$.)

Furthermore it is easy to recognize which case holds for any pair of adjacent rays if the *minimum infeasible index* defined by

$$minf(r) = \begin{cases} m+1 & \text{if } r \text{ is feasible for } P \\ min\{k : A_k\, x < 0, 1 \leq k \leq m\} & \text{otherwise} \end{cases}$$

is stored for each ray r. Clearly, case (a) occurs if and only if $minf(r) = minf(r') = m+1$, case (b) occurs if and only if $minf(r) = minf(r') = k$ for some $k \leq m$, and case (c) occurs if and only if $k = minf(r) < minf(r')$ for some $k \leq m$. Moreover, a new ray will be created if and only if

($\mathbf{c}_k$) $k = minf(r) < minf(r')$ and $k \notin Z(r')$.

To incorporate the above observations, we prepare the linked list L_k of pairs of adjacent rays which are to generate new rays at each iteration $k \le m$. We must make sure that at the beginning of a general iteration i adding the ith inequality, the list L_i contains exactly those pairs of adjacent rays that create new rays. We know that each of these rays must satisfy the condition (c_k) in some earlier iteration. In order to construct the linked list L_k successfully and free of duplications, we simply check for each candidate of adjacent rays r and r' of P^i in any earlier iteration i whether they satisfy (c_k) and it is the last chance that the condition be checked. If it is the case, store the pair in L_k.

Any pair of adjacent rays r and r' of P^i which were not adjacent in the previous cone P^{i-1} must be examined as a possible candidate for membership of L_k. Such newly-born adjacencies are of two types:

(i) the adjacency of r and r' is inherited from the previous cone P^{i-1}, that is, one of the two rays, say r, is old and the other is newly created by r and another old ray $r'' \in P^{i-1} \setminus P^i$ adjacent to r in P^{i-1} (i.e. the pair (r, r'') is in the linked list L_i);

(ii) the adjacency of r and r' is new, that is, both rays lie on the hyperplane $H_i^0 = \{x : A_i\, x = 0\}$ (i.e. $i \in Z(r) \cap Z(r')$), and either

- **(iia)** at least one of the rays is newly created; or
- **(iib)** both are old but they were not adjacent in P^{i-1}.

This classification of newly-born adjacencies is illustrated in Fig. 1 : the pairs (i, e), (j, g) are of type (i), the pairs (i, j), (i, a), (j, c) of type (iia) and (a, c) of type (iib). The enumeration of the pairs of type (i) is very simple by using the linked list L_i, but that of type (ii) is not trivial. We must take all pairs of rays lying on H_i^0 and check their adjacency.

In the algorithm to be described below, we store not only each newly created ray but also its minimum infeasible index. A procedure ConditionalStoreEdge() simply checks whether a new pair of adjacent rays satisfies the condition (c_k) for the last time, and stores the adjacency if it is the case, so that the lists L_k's are free of duplicates.

```
procedure ConditionalStoreEdge(r,r',k,i);
begin
    if there is no index i' with i < i' < k with i' ∈ Z(r) ∩ Z(r') then
        if r and r' are adjacent then
            store the pair (r,r') in the linked list L_k
        endif
    endif
end.
```

```
    procedure DDMethodVariation1(A);
    begin
        Obtain any initial DD pair (A_K, R) with R being minimal;
        Permute row indices so that K = {1, 2, ... s};
        Initialize all linked lists L_i (i = s + 1, ..., m);
        for each pair (r, r') of adjacent rays in R satisfying (c_k) do
            ConditionalStoreEdge(r,r',k,s)
        endfor;
        for i = s + 1 to m do
(1)         Obtain a DD pair (A_{K+i}, R');
                /* by generating the new rays for each pair in the linked list L_i */
(2)         for each pair (r, r') of rays in R' lying in H_i^0 and satisfying (c_k) do
                ConditionalStoreEdge(r,r',k,i);
            endfor;
            R := R';    K := K + i;
        endfor;
        Output R
    end.
```

This new variation is almost optimal in the sense that the generation of new rays in (1) is done as efficiently as possible; it is linear time. The only inefficiency, if it exists, comes from the line (2) to list all new pairs of adjacent rays. Our computational experiments indicate considerable speed-up of this implementation over the standard algorithm. See Section 5.3.

3.3 Another implementation without prefixed row ordering

What makes the combinatorial adjacency test rather expansive in DDMethod-Standard is that checking point (b) of Proposition 7 is linear in the number of extremal rays of the current cone P^i. One way to try to improve its efficiency is to first execute some faster tests to detect non adjacency. Such tests may be derived from the two following necessary conditions for adjacency and take advantage of the fact that computing the cardinality of a subset is relatively cheap.

Let $K(i)$ be the set of row indices considered in the first i iterations, and let $P^i = P(A_{K(i)})$. From point (b) of Proposition 7 we get

Proposition 9 (NC1). *A necessary condition for a pair (r, r') being adjacent in P^i is that*

$$|Z(r) \cap Z(r') \cap K(i)| \geq d - 2.$$

From the property described in the previous section of adjacency being either inherited or new, we get another, usually stronger necessary condition.

Let $b(r)$ be the birthindex of extremal ray r, that is the iteration i in which it was created. Let also $f(r)$ be the father of r, defined as the element of the pair (r', r'') from which r was created that was feasible with respect to the separating hyperplane.

Proposition 10 (NC2). *Let r and r' be distinct extreme rays of P^{i-1} separated by the hyperplane introduced at iteration i, $b(r)$ and $b(r')$ their birthindices, and $k = max\{b(r), b(r')\}$.*

Then, if the pair (r, r') did not inherit adjacency from P^{k-1}, a necessary condition for it to be adjacent in P^{i-1} is that at least one hyperplane introduced in some iteration l with $k \leq l < i$ contain both r and r', that is

$$Z(r) \cap Z(r') \cap (K(i-1) - K(k-1)) \neq \emptyset.$$

In order to implement this test we have to store for each ray its birthindex $b(r)$ and a pointer to its father $f(r)$ to recognize inherited adjacency: a pair of extreme rays of P^{i-1} has inherited adjacency if and only if one element is the father of the other.

procedure DDMethodVariation2(A);
begin
    Obtain any initial DD pair (A_K, R) for with R being minimal;
    while $K \neq \{1, 2, \ldots, m\}$ **do**
        Select i in $\{1, 2, \ldots, m\} \setminus K$ and partition J in $J^+ \cup J^0 \cup J^-$;
        /*Obtain a DD pair (A_{K+i}, R') */
        $R' := \{r_j : j \in J^+ \cup J^0\}$;
        for each pair $(r_j, r_{j'})$ with $j \in J^+$ and $j' \in J^-$ **do**
            if $(r_j, r_{j'})$ has inherited adjacency **then**
                add $r_{jj'}$ to R'
            else
                if NC2($(r_j, r_{j'})$) **then**
                    if NC1($(r_j, r_{j'})$) **then**
                        if Adj($(r_j, r_{j'})$) **then**
                            add $r_{jj'}$ to R'
                        endif
                    endif
                endif
            endif
            $R := R'$; $K := K + i$;
    endwhile;
    Output R
end

Observe that, for a fixed ordering of the rows of A, the two proposed variations differ in two aspects: the pairs that are scanned, and the iteration in which the combinatorial adjacency test is executed. Variation1 scans all the pairs lying in the current hyperplane while Variation2 scans all the pairs separated by that hyperplane. The iteration in which the combinatorial test is performed is that of the last hyperplane containing both rays in Variation1, that preceding the iteration of the separating hyperplane in the later. However, similar operations are executed in both Variations, namely testing if a small subset of the zero set of a pair is void for conditional storage in the first or for Proposition 10 in the second, and the combinatorial test is executed for exactly the same pairs when test NC1 is introduced in DDMethodVariation1.

4 Decomposition into Smaller Subproblems

For large problems, in which both the number of rows of A and the expected number of extreme rays are large, the computing time may become prohibitive (the running time for one iteration grows like the cube of the number of extreme rays in the actual cone). A natural approach to solve such a problem is to decompose it into smaller ones, and a nice feature of the Double Description method is its flexibility to handle such cases. The two following paragraphs describe simple modifications of the basic method to enumerate the extremal rays in each facet of the given cone and to preserve partial results for successive computations.

4.1 Row Decomposition

A natural way to decompose the ray enumeration problem for P is to solve the extreme ray enumeration problem for each cones $P(i)$ associated with P defined by

$$P(i) = \{x \in \mathbb{R}^d : A_i\, x = 0 \text{ and } A_k\, x \geq 0 \quad \forall k = i+1, i+2, \ldots, m\}.$$

Since the subproblem on $P(i)$ has fewer inequalities than the original problem and has one equality constraint, the extreme ray enumeration might be much easier for $P(i)$. Also, for each $P(i)$, if we output only those rays r that satisfy the ignored inequalities with strict inequality, that is, $A_1\, r > 0, \ldots, A_{i-1}\, r > 0$, then each extreme ray of the original cone will be output exactly once. Furthermore, each subproblem can be solved independently and perhaps in parallel.

The problem on $P(i)$ may be solved using the equation to eliminate one variable, but this might destroy some nice structure of A. A better way to treat equalities in the Double Description method is to modify the algorithm as follows: If i is an iteration corresponding to an equality, the iteration is performed as usual and at the end of the iteration all extremal rays which do not satisfy the equality (i.e. the rays which are strictly feasible with respect to this hyperplane) are eliminated. Then, by construction, all rays created during further iterations will still satisfy the equality.

Although in theory this row decomposition process can be applied recursively, a difficulty may occur, namely the rows of A which are redundant in $P(i)$ should be eliminated in order to make the enumeration problem really easier on $P(i)$ than on P. Redundancy of a row can be checked by solving a linear program, but repeating such a procedure will hardly be efficient. In practice, when P has a nice structure, faster heuristic or exact methods may exist to check redundancy, but the facets of P will hardly inherit this structure.

Another related technique is to enumerate extreme rays facet by facet. Unlike the row decomposition technique above, we do not eliminate already considered inequalities for each ith subproblem. Such a technique might become more efficient when P has many symmetries. In such a case, appropriately choosing one representative in each class of facets may significantly decrease the whole computing time. This occurs in many combinatorial problems, specially those with variables associated to the edges of a complete graph.

4.2 Block structure

Another decomposition, a kind of column decomposition, can be used when the matrix A has a special structure. This special structure, which we call block structure, appears frequently in combinatorial problems.

Let $i_1, \ldots, i_j, \ldots, i_d$ the indices of the first linearly independent rows of A, i.e. A_{i_j} is the first row which is not a linear combination of $A_{i_1}, \ldots, A_{i_{j-1}}$. We say that a matrix A has a block structure if

(i) for each j, all the rows A_i with $i_j \leq i < i_{j+1}$ are linear combination of $A_{i_1}, \ldots, A_{i_j}$ with nonzero coefficient of A_{i_j},

(ii) for each j and each i with $i_j \leq i < i_{j+1}$ the elements a_{ik} of the row vector A_i satisfy $a_{ik} = 0$, $\forall k > j$.

A block structure has the following interesting property. Let $\hat{P}^j$ be the cone defined by the first j columns and the first $i_{j+1} - 1$ rows of A. Then $\hat{P}^j \subseteq R^j$ and, as the system $A\,x \geq 0$ is irredundant, $\hat{P}^j$ is the projection of $\hat{P}^{j+1}$ on the hyperplane $x_{j+1} = 0$. Thus the number of extreme rays of $\hat{P}^j$ is not greater than the one of $\hat{P}^{j+1}$.

If an ordering of the rows and columns of the input matrix exists producing such a block structure, it should be maintained through the iterations of the algorithm, since it provides some control points on the number of extreme rays of the actual cone: for each successive dimension j, at the end of iteration $i_{j+1} - 1$ the number of extreme rays cannot exceed the number of extreme rays of P.

Observe that there always exists an ordering of the rows satisfying property (i) and having thus a projection property. The block structure is a special case of such an ordering, where the projection property holds for each successive dimension and where the cones $\hat{P}^j$ generally have a direct interpretation (typically the solution of the main problem on a subgraph).

We now describe how to modify the basic double description algorithm in order to maintain the block structure, that is to work in a general iteration i on a nonpointed cone.

Let $P^{i-1} = \{x \in \mathbb{R}^d : A_k\ x \geq 0 \text{ for } k = 1, \ldots, i-1\}$ be described by a minimal set of generators (R, B) such that $P^{i-1} = \{x \in \mathbb{R}^d : x = R\lambda + B\mu \text{ for some } \lambda \geq 0 \text{ and some } \mu\}$, where $B = \{b_1, \ldots, b_t\}$ is a basis of the lineality space and $R = \{r_1, \ldots, r_s\}$ a set of representatives of the minimal proper faces (faces of dimension t+1) of P^{i-1}.

Then a minimal set of generators (R', B') of $P^i = \{x \in \mathbb{R}^n : A_k\ x \geq 0 \quad \forall k = 1, \ldots, i\}$ is determined by the following rules:

a) if $A_i \perp B$, then $B' = B$ and R' is obtained as stated in Lemma 8 , two generators r_j and $r_{j'}$ in R being adjacent if they are contained in a $(t+2)$-dimensional face of P^{i-1}.

b) else choose $\hat{B} = \{b'_1, \ldots, b'_t\}$ such that $A_i \cdot b'_k = 0$ for $k = 1, \ldots, t-1$ and $A_i \cdot b'_t > 0$, and set $B' = \{b'_1, \ldots, b'_{t-1}\}$ and $R' = \{r'_1, \ldots, r'_s, b'_t\}$, with $r'_j = (A_i \cdot b'_t)\, r_j - (A_i \cdot r_j)\, b'_t \quad$ for $j = 1, \ldots, s$.

Initializing the algorithm with $P^0 = \{x \in \mathbb{R}^d : x = B^0\mu$ for some $\mu\}$ and $B^0 = A_{LS}^{-1}$ where $A_{LS} = \{A_{i_j}\ , j = 1, \ldots, d\}$ avoids the computation of $\hat{B}$ in case b) as the vectors of B^0 already satisfy the required property. When the matrix A has the block structure, B^0 may be chosen as the canonical basis of $\mathbb{R}^d$ and the iterations i_j are particularly easy to perform since only the sign of the jth unit vector has to be properly chosen.

Note also that after an iteration i with $A_i \not\perp B$ each pair (r'_j, b'_t) , $j = 1, \ldots, s$ is adjacent in P^i.

A block structure is not only part of a good ordering, it can also often be used in conjunction with row decomposition to reduce the computing time: if h is the last iteration completed on the main problem P and the generators of P^h have been saved, the computation of each subproblem corresponding to a facet $P \cap H_i^0$ may be started at iteration $h+1$ with either P^h if $i > h$ or those generators of P^h lying in H_i^0 if $i \leq h$.

5 Computational Results

In this section we present some of the experiments we carried out with the different implementations of the DD Method. The test examples are part of the distribution of CDD and thus publically available by anonymous ftp [Fuk93]. The input matrices A come from various combinatorial, geometric and practical applications, namely the vertices of the hypercube (cube) or its dual (cross), the complete cut cone (ccc) or cut polytope (ccp), the arborescences - rooted directed trees - in small graphs (prodmT), the vertices of a regular polytope (reg), a ternary alloy ground state analysis (mit). All cones except the hypercube are degenerate.

5.1 Experiments on Adjacency Tests

Table 1 contains the running times of algebraic and combinatorial adjacency test on some examples. It shows that the combinatorial test is almost always faster. A partial explanation for that is the following: One can expect the algebraic test to perform better for proving adjacency (exhibit $d-2$ linearly independent rows) and the combinatorial for proving nonadjacency (exhibit one extreme ray violating Proposition 7), but the number of tests executed on nonadjacent pairs is usually much larger.

Table 1. Comparison of Algebraic and Combinatorial Adjacency Tests

test problems	sizes			Algebraic Test	Combin. Test
	dim	m	#output	(in seconds)	(in seconds)
reg24-5	4	24	24	7	4
reg600-5	4	600	120	695	523
cross8	8	256	16	189	45
ccc5	10	15	40	9	6
ccp5	10	16	56	17	7
cube10	10	20	1,024	87	25
ccc6	15	31	210	903	101
ccp6	15	32	368	1,663	179

(On Mac IIci running Mach OS; cdd-038 compiled by gcc-2.6.0)

5.2 Experiments on Row Ordering

Different static and dynamic strategies for row ordering are compared in Table 2. The dynamic strategies used consist in selecting an hyperplane that cuts a maximum (minimum) number of the actual extremal rays. Except for the hypercube, for which all strategies perform equally well, the lexicographic ordering is always better, dominating mincutoff, then random and maxcutoff. For problems with a large number of rows in higher dimension a lexicographic ordering or some variant (obtained for example by permuting the columns) is the only useful one.

Table 2. Comparison of Different Row Orderings

test problems	sizes			max intermediate size (average size)			
	dim	m	#output	random	lexmin	mincutoff	maxcutoff
reg600-5	4	600	120	494 (319)	168 (94)	187 (134)	444 (276)
cross8	8	256	16	813 (272)	22 (17)	293 (128)	948 (270)
cube10	10	20	1,024	1024 (210)	1024 (231)	1024 (209)	1024 (231)
ccc6	15	31	210	757 (372)	575 (186)	575 (248)	752 (391)
ccp6	15	32	368	1484 (752)	693 (257)	1092 (441)	1682 (739)

(Solved with Variation1 (cdd-055a))

5.3 Experiments on the New Edge Data Updates

Table 3 shows the speedup which can be obtained with the improvements on the standard method. Clearly these ratios depend also on the dimensions and the ordering used.

Table 3. Comparison of DD Standard and Variation1

test problems	sizes			DDStandard	DDVariation1	speedup ratio
	dim	m	#output	(in seconds)	(in seconds)	(times faster)
reg600-5	4	600	120	13	5	2.60
mit729-9	8	729	4,862	42,587	5,959	7.15
cube12	12	24	4,096	15	5	3.00
cross12	12	4,096	24	111	89	1.25
cube14	14	28	16,384	238	69	3.45
ccc6	15	31	210	2	2	1.00
ccp6	15	32	368	5	3	1.67
cube16	16	32	65,536	4,812	1,512	3.18
prodmT5	19	711	76	39	25	1.56
ccc7	21	63	38,780	152,694	142,443	1.07
ccp7	21	64	116,764	621,558	469,681	1.32
prodmT62	24	3,461	168	15,083	10,047	1.50

(Standard(cdd-038) and Variation1(cdd-055a) compiled by gcc-2.6.2 on Spark-Server 1000)

5.4 Comparison of DD Variation1 and Variation2

Relative performances of Variation1 and 2 are presented in Table 4. None of these variations dominates the other, but Variation1 is usually slightly faster since there are less pairs lying in a common hyperplane than pairs separated.

Table 4. Comparison of DDVariation1 and DDVariation2

test problems		reg600-5	prodmT62	ccp6	cross14	cube14
dimensions(A)		600x5	3,461x25	32x16	16,384x15	28x15
#output		120	168	368	28	16,384
DDV1	#pairs scanned	24,386	25,021,758	563,086	2,728,111	89,494,867
	#pairs separated	12,443	5,041,661	125,643	372,827	16,382
	time(sec)	3	1,038	4	560	72
DDV2	#pairs scanned	512,257	19,723,402	168,038	196,610	16,382
	time(sec)	9	1,158	5	544	3
both	#comb.tests	1,304	2,091,514	4,691	17,327	0
	#rays (max)	524	1,146	693	40	16,384
	#rays (total)	1,889	65,239	1,768	16,504	16,384

(Variations 1 and 2 coded in pascal and run on Silicon Graphics)

References

[AB95] D. Avis and D. Bremner. How good are convex hull algorithms. Technical Report TR95.1, School of Computer Science, McGill University, Montreal, Canada, 1995. available via anonymous ftp from mutt.cs.mcgill.ca (directory pub/doc).

[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 and qrs available via anonymous ftp from mutt.cs.mcgill.ca (directory pub/C).

[BDH93] C.B. Barber, D.P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. Technical Report GCC53, The Geometry Center, Minnesota, U.S.A., 1993.

[CGAF94] G. Ceder, G.D. Garbulsky, D. Avis, and K. Fukuda. Ground states of a ternary fcc lattice model with nearest and next-nearest neighbor interactions. *Physical Review B*, 49(1):1–7, 1994.

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

[DFL93] M. Deza, K. Fukuda, and M. Laurent. The inequicut cone. *Discrete Mathematics*, 119:21–48, 1993.

[Dye83] M.E. Dyer. The complexity of vertex enumeration methods. *Math. Oper. Res.*, 8:381–402, 1983.

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

[Fuk93] K. Fukuda. *cdd.c : C-implementation of the double description method for computing all vertices and extremal rays of a convex polyhedron given by a system of linear inequalities.* Department of Mathematics, Swiss Federal Institute of Technology, Lausanne, Switzerland, 1993. program available from ifor13.ethz.ch (129.132.154.13), directory /pub/fukuda/cdd.

[Grü67] B. Grünbaum. *Convex Polytopes.* John Wiley and Sons, New York, 1967.

[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.

[MS71] P. McMullen and G.C. Shephard. *Convex Polytopes and the Upperbound Conjecture.* Cambridge University Press, 1971.

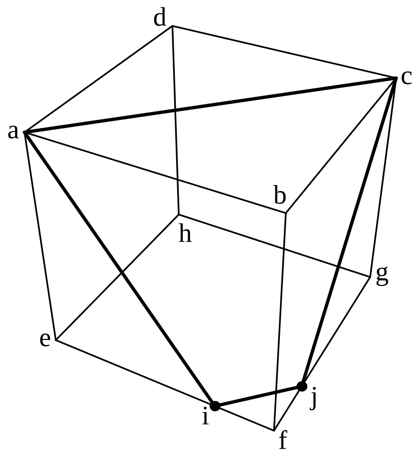
[Mul94] K. Mulmuley. *Computational Geometry, An Introduction Through Randamized Algorithms.* Prentice-Hall, 1994.

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

[Ver92] H. Le Verge. A note on Chernikova's algorithm. Technical report internal publication 635, IRISA, Rennes, France, February 1992.

[Wil93] D. Wilde. A library for doing polyhedral operations. Technical report internal publication 785, IRISA, Rennes, France, December 1993.

This article was processed using the LaTeX macro package with LLNCS style


d
c
a
b
h
g
e
j
i
f