lrs: A Revised Implementation of the Reverse Search Vertex Enumeration Algorithm

David Avis

School of Computer Science
McGill University
3480 University, Montréal, Québec, Canada H3A 2A7
avis@cs.mcgill.ca

January 26, 1999

ABSTRACT

This paper describes an improved implementation of the reverse search vertex enumeration/convex hull algorithm for d-dimensional convex polyhedra. The implementation uses a lexicographic ratio test to resolve degeneracy, works on bounded or unbounded polyhedra and uses exact arithmetic with all integer pivoting. It can also be used to compute the volume of the convex hull of a set of points. For a polyhedron with m inequalities in d variables and known extreme point, it finds all bases in time $O(md^2)$ per basis. This implementation can handle problems of quite large size, especially for simple polyhedra (where each basis corresponds to a vertex and the complexity reduces to $O(md)$ per vertex). Computational experience is included in the paper, including a comparison with an earlier implementation.

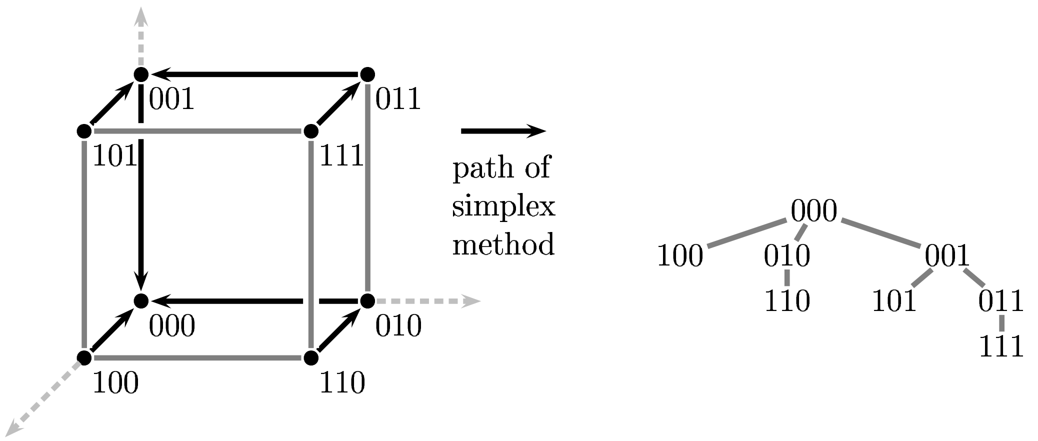
1. Introduction

This paper describes *lrs* [1], a revised version of the reverse search vertex enumeration algorithm proposed by Fukuda and the author [2]. This implementation is a major improvement on *rs*, a program released by the author in 1992, and revised in 1994 [4], which was based on the original method described in [2]. The improvements in speed are attributable to many factors, including an improved pivot selection procedure to resolve degeneracy lexicographically, faster arithmetic based on integer rather than rational arithmetic, and optional cacheing to reduce backtrack pivots. The new implementation is more general, as it handles bounded and unbounded polyhedra, does not require non-negativity of variables, and performs automatic transformations to allow solution of facet enumeration and Voronoi diagram problems.

The main function of *lrs* is to find the vertices and extreme rays of a polyhedron described by a system of linear inequalities. A description of the theory underlying the current implementation of this function is the purpose of this paper. Additional functions of *lrs* are built on top, and are described briefly here. These include: facet enumeration, computation of Voronoi vertices, volume computation, estimation of the output size, and restart capability. The remainder of this introduction contains an informal description of how *lrs* works. Section 2 of the paper gives some background material and basic definitions. Section 3 describes dictionaries, their relationship to vertices and extreme rays and pivoting. Section 4 describes lexicographic pivoting and proves that each vertex and extreme ray is representable by a lex-positive basis. Section 5 is concerned with the unique representation of vertices and extreme rays by lex-min bases. Section 6 contains a description of reverse search, and an efficient implementation of the reverse function as used in *lrs*. Section 7 describes some implementation issues, including the integer pivoting formulae used. Section 8 describes briefly how the additional functions mentioned above are implemented. Section 9 concludes with some limited computational experience comparing the current version *lrs* to the earlier version *rs*. More extensive experience, and an empirical analysis of the various speedups is contained in Avis[7].

Briefly and informally, the reverse search algorithm works as follows. Suppose we have a system of m linear inequalities defining a d-dimensional polyhedron in R^d and a vertex of that polyhedron given by the indices of d inequalities whose bounding hyperplanes intersect at the vertex. These indices define a *cobasis* for the vertex. The complementary set of $m-d$ indices are called a *basis*. For any given linear objective function, the simplex method generates a path between *adjacent* bases (or equivalently cobases) which are those differing in one index. The path is terminated when a basis of a vertex maximizing this objective function is found. The path is found by pivoting, which involves interchanging one of the hyperplanes defining the current cobasis with one in the basis. The path chosen from the initial given basis depends on the pivot rule used, which must be finite to avoid cycling. The original implementation *rs* used Bland's least subscript rule [8]. If we look at the set of all such paths from all bases of the polyhedron, we get a spanning forest of the graph of adjacent bases of the polyhedron. The root of each subtree of the forest is a basis of an optimum vertex. The reverse search algorithm starts at each root and traces out its subtree in depth first order by *reversing* the pivot rule.

The algorithm is particularly easy if the polyhedron is *simple* (non-degenerate): each vertex lies on exactly d hyperplanes, and so has a unique basis. The spanning forest has one component, which is a spanning tree of the skeleton of the polyhedron, and each vertex is produced once. An example of such a polyhedron is the cube, and Figure 1.1 shows a possible reverse search tree for it.



(a) The "simplex tree" induced by the objective $(-\sum x_i)$.

(b) The corresponding reverse search tree.

Figure 1.1

A complication of the original implementation was the need to handle a degenerate starting vertex. A dual form of Bland's rule had to be implemented in order to compute all bases of this vertex. Each of these bases was then used as the start for a (primal) reverse search. A serious drawback of the algorithm as originally implemented, was that it computed all bases of the input polyhedron. For many polyhedra encountered, especially combinatorial polytopes, the number of bases is much larger than the number of vertices.

The standard approach to reducing the number of bases is perturbation: make small changes to the input data so that the resulting polyhedron is simple. The resulting perturbed polyhedron will typically have more vertices, but far fewer bases, than the original polyhedron. Numerical perturbation was implemented for *rs*, but abandonded because: (i) perturbed vertices have to be transformed back to give true vertices; (ii) the answer may no longer be correct as vertices may be lost; (iii) perturbation increases the cost of the extended precision arithmetic; (iv) a vertex is output more than once.

The current version *lrs* resolves degeneracy by use of the well-known lexicographic pivot selection rule for the simplex method. This rule is defined for a subset of the bases, known as lex-positive. The subgraph of lex-positive bases forms a connected subgraph of the basis graph which covers all vertices of the polyhedron. Furthermore an objective function can be chosen so that the simplex method initiated at any lex-positive basis terminates at a unique lex-positive optimum basis. If we initiate the reverse search method at this basis and reverse the lexicographic pivot rule we generate a spanning tree of the graph of all lex-positive bases. This is the core of *lrs*.

As we will see in Section 5, the lex-min basis for each vertex is lex-positive. We can test the property of being a lex-min basis quickly, and report only lex-min bases. Lexicographic pivoting is sometimes referred to as symbolic perturbation, since the graph of lex-positive bases can be viewed as the skeleton of a simple polyhedron obtained by perturbation of the original polyhedron. Indeed, it is often described by adding powers of a small indeterminate ε to the right hand side of each inequality to resolve degeneracy. However it can be implemented without any change to the original data, and so all of the objections (i)-(iv) above are resolved. Furthermore we can use the same method for the dual problem of computing the facets of the convex hull of as set of vertices and extreme rays. In this dual context, properly interpreted, the graph of lex-positive bases corresponds to a triangulation of the convex hull. Since the determinant of each of these simplices is known (see Section 7), an additional advantage is that we can obtain the volume of a polytope given by a set of vertices for the same cost as computing the facets of its convex hull.

A remarkable feature is that no additional storage is needed at intermediate nodes in the tree. Going deeper in the tree we explore all valid "reverse" pivots in order by basic index from any given intermediate node. For backtracking, we can use the pivot rule to return us to the parent node along with the current pivot indices. From there it is simple to continue by considering the next basic index as a "reverse" pivot, etc. The algorithm is therefore non-recursive and requires no stack or other data structure. The output is produced as a duplicate free stream (see Section 5), which may be useful even if the computation is too large to complete. Empirically, about 80-90% of the running time of *lrs* is spent pivoting. Since backtracking generates exactly half of the pivots, considerable savings can however be obtained by "cacheing" dictionaries along the current path from the root.

Although *lrs* is a large improvement on *rs*, it is far from an efficient general solution to the vertex enumeration problem. Such a solution should reasonably be required to generate all vertices in time polynomial in the input and output size. Currently no such algorithm is known to exist. Examples contained in Avis, Bremner and Seidel [6] show that all pivot algorithms using numeric or symbolic perturbation may behave extremely badly: the number of bases computed can be super-polynomial in the number of vertices. This is born out in practice for combinatorial polytopes. *lrs* is efficient for vertex enumeration of simple (or near-simple) polyhedra, or dually for facet enumeration of simplicial (or near-simplicial) polyhedra. Recently Bremner, Fukuda and Marzetta [9] developed an ingenious primal-dual method for vertex enumeration of highly degenerate simplicial polytopes that satisfy a hereditary property. It works by simulating the reverse search tree generated by *lrs* for the (easy) dual facet enumeration problem for simplicial polytopes. Dually, this method can be used for facet enumeration of simple polytopes. For polytopes with zero-one vertices, a polynomial time vertex enumeration algorithm was recently announced by Bussieck and Luebbecke [12].

lrs can be efficiently parallelized, as has been done by Brüngger, Marzetta, Fukuda and Nievergelt [10]. This parallel version has been used to solve some extremely large problems which do not seem solvable by other methods.

2. Background

For a general introduction to convex polyhedra, the reader is referred to Ziegler [20]. Throughout the paper we will assume that P is a d-dimensional polyhedron in R^d. A classic result is that P can be represented in two ways. An *H-representation* is given by a $m \times d$ matrix

$\bar{A} = (\bar{a}_{i,j})$ and a m-vector $\bar{b} = (\bar{b}_i)$:

$$P = \{ y \in R^d \mid \bar{b} + \bar{A} y \geq 0 \} \tag{2.1}$$

If $\bar{A}$ is minimal, that is no row can be deleted without changing P, then P has m *facets*, each defined by one of the inequalities in (2.1). A *vertex* $y \in R^d$ is a point of P that satisfies an affinely independent set of d inequalities as equations. We assume throughout that P has at least one vertex, which implies that $m \geq d + 1$. An *extreme ray* $z \in R^d$ is a direction such that for some vertex y and any positive scalar t, $y + tz$ is in P and satisfies some set of $d - 1$ affinely independent inequalities as equations. Note that an extreme ray is unique only up to a positive scalar, since if z is an extreme ray then so is tz for any positive scalar t. An equivalent *V-representation* of P is given by a minimal set of *s vertices* $y_1, \cdots, y_s$ and *u extreme rays* $z_1, \cdots, z_u$:

$$P = \{ y \in R^d \mid y = \sum_{i=1}^{s} \lambda_i y_i + \sum_{j=1}^{u} \mu_j z_j, \lambda_i \geq 0, \mu_j \geq 0, \sum_{i=1}^{s} \lambda_i = 1 \}. \tag{2.2}$$

The *vertex enumeration problem* is to produce a V-representation from an H-representation, and the *facet enumeration problem* is to provide the reverse transformation. It is well known that these problems are essentially equivalent, see Section 8.1. In this paper we treat the problem primarily from the vertex enumeration perspective.

As an example, consider the unbounded three dimensional polyhedron P defined by the system of 8 inequalities:

$$\begin{aligned} 1 + x_1 &\geq 0 \\ 1 + x_2 &\geq 0 \\ 1 - x_1 &\geq 0 \\ 1 - x_2 &\geq 0 \\ 1 - x_1 + x_3 &\geq 0 \\ 1 - x_2 + x_3 &\geq 0 \\ 1 + x_1 + x_3 &\geq 0 \\ 1 + x_2 + x_3 &\geq 0 \end{aligned}$$

It has a V-representation given by the 5 vertices

$$(1, 1, 0), \ (-1, 1, 0), \ (1, -1, 0), \ (-1, -1, 0), \ (0, 0, -1)$$

and one extreme ray (0,0,1). Note that the vertices do not have unique cobases. For example the vertex (-1,-1,0) can be defined by choosing any three of the first two and last two inequalities and replacing them by equations. The extreme ray has four representations, one with each of the first four vertices. For example $(1, -1, 0) + (0, 0, t)$ satisfies both the second and third inequalities as equations, and is in P for all $t > 0$. These representations with distinct vertices are called *geometric rays*.

3. Dictionaries

Much of the material in this section is adapted to the vertex enumeration setting from standard results in linear programming, see for example Chvátal [13] and Ignizio and Cavalier [18]. As with the simplex method, the essential calculations are performed on a *dictionary* derived from (2.1). We distinguish *decision* variables $x_1, \ldots, x_d$ and *slack* variables $x_{d+1}, \ldots, x_{d+m}$. It is easy to see that solutions to (2.1) can be put in 1-1 correspondence to solutions of

$$x_{d+i} = \bar{b}_i + \sum_{j=1}^{d} \bar{a}_{i,j} x_j, \qquad i = 1, .., m.$$

$$x_{d+i} \geq 0, \qquad i = 1, \ldots, m$$

by identifying $x_j = y_j$, $j = 1, \ldots, d$. *lrs* is initiated from a vertex of P, which is either supplied by the user or is computed by solving a linear program over (2.1). It is convenient to order the rows of (2.1) so that the final d rows define this initial vertex. Since these rows are affinely independent, we can rewrite the above system of m equations as the equivalent system

$$x_i = b_i + \sum_{j=1}^{d} a'_{i,j} x_{m+j}, \qquad i = 1, \ldots, m. \tag{3.1}$$

for a suitable coefficients b_i and $a'_{i,j}$. For reasons that become clear later, we augment this system by adding the additional equation

$$x_0 = b_0 + \sum_{j=1}^{d} a'_{0,j} x_{m+j} \tag{3.2}$$

where $b_0 = 0$ and $a'_{0,j} = -1, j = 1, \ldots, d$. From the vector $b = (b_0, \ldots, b_m)$ and $(m+1) \times d$ matrix $A' = (a'_{i,j})$ we form the $(m+1) \times (d+m+1)$ dimensional matrix

$$A = [\ I \quad -A'\ ]$$

where I is an $(m+1) \times (m+1)$ identity matrix. Then the augmented system of equations can be rewritten as

$$Ax = b. \tag{3.3}$$

For any matrix such as A, A_j refers to the j-th column of A, and A_J refers to the submatrix of columns of A indexed by J. We use similar notation for vectors. The notation A^i and A^I refers to row i and the submatrix of A with rows indexed by I respectively. Let B be an ordered *m+1-tuple* indexing a set of $m+1$ affinely independent columns of A, and let N be an ordered *d-tuple* indexing the remaining d columns. Then we may rewrite (3.3) as $A_B x_B + A_N x_N = b$, which is equivalent to the system:

$$I\ x_B + A_B^{-1} A_N x_N = A_B^{-1} b \tag{3.4}$$

The system (3.4) is called a *dictionary*, and consists of $m+1$ rows. The t-th row corresponds to the t-th index in B. All essential calculations involve manipulating this dictionary. The index set B is called a *basis*, and the index set N is called a *cobasis*. From any dictionary we obtain a *basic solution* x by setting $x_B = A_B^{-1} b$ and $x_N = 0$. B is a *feasible basis*, x is a *basic feasible solution*, and (3.4) is a *feasible dictionary* if:

(i) $\{0, \ldots, d\} \subset B$, and

(ii) $x_i \geq 0,\ i \in B,\ i = d+1, \ldots, d+m$.

In other words, B contains x_0 and all decision variables, and x is non-negative for all slack variables. The next two propositions show how the vertices and extreme rays of P can be recovered from dictionaries.

Proposition 3.1: *Every vertex y of P can be extended to a basic feasible solution of (3.4), and every basic feasible solution x of (3.4) can be restricted to a vertex of P.*

Proof: Let y be a vertex of P. It is defined by choosing a a set of d inequalities in (2.1) and replacing them by equations. Let N be the set of all indices $i+d$ such that inequality i is chosen, and let $B = \{0, \ldots, d+m\} - N$. Let $x_j = y_j, j = 1, \ldots, d$, define $x_{d+i}, i = 1, \ldots, m$ by (3.1) and x_0 by (3.2). Clearly $x_N = 0$ and (i) and (ii) are satisfied since y satisfies (2.1). Conversely, if x is a basic feasible solution, we define y by $y_j = x_j, j = 1, \ldots, d$. Since $x_N = 0$, N indicates a set of d inequalities from (2.1) that are solved as equations to define y. Condition (ii) indicates y is feasible for (2.1), hence it is a vertex. □

Proposition 3.2: *For every extreme ray z of P, there is a feasible basis B and index $s \in N$ such that letting $a = A_B^{-1} A_s$:*

(a) $z_i = -ta_i, \quad 1 \leq i \leq d$, and some $t > 0$,

(b) $a_i \leq 0, \ d+1 \leq i \leq m$

Proof: From the definition, each extreme ray of P is a feasible direction $z \in R^d$, so that for some vertex y and any positive scalar t, $y + tz$ satisfies a set of $d-1$ inequalities from (2.1) as equations. Let N' index this set. Since P does not admit lines, this ray is terminated by one additional inequality being satisfied as an equation. Let s index this inequality and set $N = N' + s$. We obtain a set of d cobasic indices N which determine the vertex y of P. Let $a = A_B^{-1} A_s$. By (b) we can obtain feasible solutions $x = A_B^{-1} b - at$ by setting $x_{N'} = 0$ and $x_s = t$ for any positive scalar t. All these solutions satisfy inequalities indexed by N' as equations, so restricting to coordinates $1, \ldots, d$ of $-a$, we obtain the extreme ray z. □

Under the conditions of Proposition 3.2, we say that B *represents* the geometric ray $y + tz$. *lrs* computes feasible dictionaries from which vertices and extreme rays of P may be obtained. The basic operation is a *pivot* between two bases B and $\bar{B}$, which is defined by indices $r \in B$ and $s \in N$ by setting $\bar{B} = B - r + s$. This notation means that s replaces r in its position (say t) in the basis B. The operation is the computation of (3.4) for $\bar{B}$, which is done as follows. Let $a = A_B^{-1} A_s$.

Pivot Operation.

(a) Divide row t of (3.4) by a_t.

(b) For $i = 0, \ldots, t-1, t+1, \ldots, m$, subtract a_i times the new row t from row i of (3.4).

We discuss the mechanics in Section 7. The pivot is *feasible* if both x_B and $x_{\bar{B}}$ are basic feasible solutions, which implies that both x_r and x_s are slack variables. The decision variables do not play any active role in pivot selection, they are "carried" along merely to enable the vertex and extreme ray coordinates to be computed (they are columns of this part of the dictionary).

Let $B^* = \{0, \ldots, m\}$ and $N^* = \{m+1, \ldots, m+d\}$. This is the initial basis and cobasis, and the initial dictionary is given by (3.2) and (3.1), which we write:

$$I \; x_{B^*} + A_{N^*} x_{N^*} = b. \tag{3.5}$$

The first equation in this system is

$$x_0 + \sum_{i \in N^*} x_i = 0 \tag{3.6}$$

If we interpret x_0 to be the value of an objective function, then the basic solution of this dictionary with $x_{N^*} = 0$ has objective value zero. Since for all basic feasible solutions we have $x_j \geq 0,\ j \in N^*$, it is clear that the maximum of x_0 over all basic feasible solutions is zero, and the initial dictionary achieves this maximum. This initial dictionary is the root of the reverse search tree constructed by *lrs*. All other bases are obtained by pivoting from this dictionary. The initial basis inverse is the $(m+1) \times (m+1)$ identity matrix. To perform lexicographic pivots from any feasible basis B, it will be necessary to have access to the basis inverse A_B^{-1}. This can be readily obtained from the dictionary (3.4) with basis B as follows. Let e^i be the unit $m+1$-vector indexed $0, \ldots, m$ with a one at index i.

Proposition 3.3: *Let B be a feasible basis. Then, for $i = 0, \ldots, m$, column i of A_B^{-1} is e^r if $i = B_r$ for some r, otherwise it is the column $A_B^{-1} A_i$.*

Proof: The dictionary (3.4) corresponding to B is obtained by standard row and column operations from the equivalent system (3.5). It follows from elementary linear algebra that the columns indexed by $B^* = \{0, \ldots, m\}$ (3.4) are the basis inverse A_B^{-1}. □

4. Lex-positive Bases and Lexicographic Pivot Selection

The earlier program *rs* computed all basic feasible solutions, and so by Proposition 3.1 this guarantees all vertices will be found. For degenerate polyhedra there are many more feasible bases than vertices, so it useful to identify a smaller set of bases that still cover all the vertices. We call a vector *lex-positive* if its first non-zero coordinate is positive. We call a feasible basis B *lex-positive* if each of the rows indexed $d+1, \ldots, , m$ of the $(m+1) \times (m+1)$ matrix

$$D = \left[\; A_B^{-1} b \;\; A_B^{-1} \; \right] = \begin{bmatrix} \beta_0 & \alpha_{0,1} & \cdots & \alpha_{0,m} \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \beta_m & \alpha_{m,1} & \cdots & \alpha_{m,m} \end{bmatrix} \tag{4.1}$$

is lex-positive. The initial basis B^* is lex-positive, since in this case (4.1) is [b I], and the condition follows from the feasibility of B^*. Every feasible basis B contains indices 0,...,d and so the columns with these indices in (4.1) remain unchanged, they are the first $d+1$ columns of the identity matrix. We will show that each vertex has a lex-positive basis. To do this we introduce lexicographic pivoting, which preserves the property that the basis is lex-positive.

Let B be a lex-positive basis. Let $s \in N$ and consider the column $a = A_B^{-1} A_s$. If $a_i \le 0, d+1 \le i \le m$, this column defines an extreme ray, as described in Proposition 3.2. Otherwise, let t be the index such that D^t / a_t is the lexicographically minimum vector of

$$\left\{ \frac{D^i}{a_i} : a_i > 0, d+1 \le i \le m \right\} \tag{4.2}$$

Such a minimum is unique, because D has full row rank. Let r be the basic index in B corresponding to row t. We abbreviate this in the notation:

$$r = lexminratio(B, s)$$

In case (4.2) is empty and we set $r = 0$.

Proposition 4.1: *Given a lex-positive basis B and $s \in N$, let $r = lexminratio(B, s) \ne 0$.*

(a) $\bar{B} = B - r + s$ is a lex-positive basis.

(b) $s = lexminratio(\bar{B}, r)$.

Proof: Let t be the row of D corresponding to basic index $r \in B$, and set $a = A_B^{-1} A_s$. The pivot operation produces the matrix $\bar{D}$ for $\bar{B}$ from D according to the formulae:

$$\bar{D}^t = \frac{D^t}{a_t}, \qquad \bar{D}^i = D^i - a_i \bar{D}^t, \qquad 0 \le i \le m, \ i \ne t. \tag{4.3}$$

To prove the lex-positivity of $\bar{B}$ we need only consider rows indexed $d+1 \le i \le m$. Since each row of D is lex-positive, and $a_t > 0$, $\bar{D}^t$ is lex-positive. Therefore, for each i with $a_i \le 0$, $\bar{D}^i$ is the sum of a lex-positive vector with either zero or a lex-positive vector, so it is lex-positive. For each i with $a_i > 0$ we can rewrite its equation above as

$$\frac{\bar{D}^i}{a_i} = \frac{D^i}{a_i} - \frac{D^t}{a_t}.$$

Since t was chosen as the lexicographically minimum vector in (4.2), it follows that the right hand side and hence $\bar{D}_i$ is lex-positive. This proves part (a). For part (b) set

$$\bar{a} = A_{\bar{B}}^{-1} A_r.$$

In the dictionary with basis B, column r is the identity column e^t. Therefore it follows from the pivot operation that

$$\bar{a}_t = \frac{1}{a_t}, \qquad \bar{a}_i = -a_i \bar{a}_t, \qquad d+1 \le i \le m, \ i \ne t.$$

Using these definitions, for any i, $d+1 \le i \le m$, $i \ne t$ such that $\bar{a}_i > 0$ we can deduce from (4.3)

$$\frac{\bar{D}^i}{\bar{a}_i} = \frac{D^i}{\bar{a}_i} + \frac{\bar{D}^t}{\bar{a}_t}.$$

Since both terms on the right hand side are lex-positive, we conclude that $\bar{D}^i / \bar{a}_i$ is lexicographically greater than $\bar{D}^t / \bar{a}_t$. Therefore row t is the minimizer for basis $\bar{B}$. This proves part (b), since $s \in \bar{B}$ is the index corresponding to row t of the updated dictionary.

□

The lexicographic pivot rule for the simplex method for linear programming chooses pivots in the following way. Assuming the problem is to maximize x_0, the index $s \in N$ of the entering variable is chosen so that it has a negative coefficient in row zero of (3.4). If no such index exists, the current basic feasible solution is optimum. We then choose $r = lexminratio(B, s)$ as the index of the leaving variable. If no such index exists, the problem is unbounded. The proof of Proposition 4.1 shows that each pivot adds a positive multiple of the lex-positive row r to the objective function. Since the objective row increases lexicographically with each pivot, this proves no basis can repeat and the optimum (or an unbounded solution) must eventually be reached. Furthermore, if pivots are chosen in this way, an optimum solution will be found with a lex-positive basis, even though in general it may have many other bases that are not lex-positive. We may now strengthen Proposition 3.1.

Proposition 4.2: *Every vertex y of P can be extended to a basic feasible solution of (3.4) with a lex-positive basis.*

Proof: It is well known that for every vertex y of P there is a linear function which obtains its maximum over P at the unique point y. We use this function instead of the the one specified in (3.2), and initiate the simplex method using lexicographic pivoting on the resulting dictionary (3.4) with the initial lex-positive basis B^*. By the above discussion, we will obtain the optimum solution y with a lex-positive basis. □

We call a feasible basis B and its corresponding dictionary *optimum* if row zero of $A_B^{-1}A$ is non-negative. As described above, this is the normal stopping criterion for the simplex method. An important feature of lex-positive bases is that there is a unique optimum dictionary, eliminating the need to initiate reverse search on multiple dictionaries. A similar result was obtained by Bremner et al.[9] using a different argument.

Proposition 4.3: *B^* is the unique optimum lex-positive basis.*

Proof: Row zero of the initial dictionary is the $m + d + 1$ vector (1,0,...,0,1,...,1) with m zeroes, derived from equation (3.6), where the cobasis is $N^* = \{m + 1, \cdots, m + d\}$. Suppose there exists another lex-positive optimum basis $B \neq B^*$. Let $k_1, \ldots, k_q$ be indices in $B - B^*$. Each index is necessarily in N^*. Consider row zero of the system (3.4). It is obtained from (3.6) by subtracting equations with variables indexed by $k_1, \ldots, k_q$ from (3.4), since these variables are eliminated from the cobasis N^*. The coefficients of variables $x_0, \ldots, x_m$ in these equations are the corresponding rows in (4.1), each of which is lex-positive by assumption. Since both B and B^* are optimum, $\beta_{k_j} = 0$. It follows that $(\alpha_{0,1}, \cdots, \alpha_{0,m})$ is lexicographically negative, so $\alpha_{0,j} < 0$ for some index $d + 1 \leq j \leq m$ (in fact, the first non-zero coefficient). This contradicts the optimality of B. □

The results of this subsection are summarized in the following theorem.

Theorem 4.4: *For each vertex y of P there is a lex-positive basis B with basic feasible solution x such that $x_i = y_i, i = 1, \ldots, d$. The simplex method initiated on the corresponding dictionary (3.4) generates a sequence of lex-positive bases terminating in the basis B^*.* □

5. Lex-minimum Bases of Vertices and Extreme Rays

Even with lex-positive pivoting, for a non-simple polyhedron the same vertex will often be generated with many bases. Fortunately, a unique representative basis can easily be identified for each vertex. For a given basic feasible solution x of (3.4), its lexicographically smallest basis B is called *lex-min*.

Proposition 5.1: *B is lex-min for some basic feasible solution x if and only if there does not exist $r \in B$ and $s \in N$ such that (i) $r > s$, (ii) $x_r = 0$, and (iii) $(A_B^{-1} A_s)_r \neq 0$.*

Proof: Clearly if B is lex-min we cannot satisfy (i)-(iii), for otherwise, the basis $\bar{B} = B - r + s$ has the same basic feasible solution x and is lexicographically smaller than B. On the other hand, suppose B is not lex-min. Let $B_{\min}$ be the lex-min basis with the same basic feasible solution x. Note that this implies that if $x_i \neq 0$, i is contained in both B and $B_{\min}$. Let s be the smallest index in $B_{\min} - B$. Since B is a basis there is some index $r \in B - B_{\min}$ such that $\bar{B} = B - r + s$ is a feasible basis. By the choice of r and s we must have both (i) and (ii). Finally (iii) follows from the fact that r and s define a valid pivot for the dictionary with basis B. □

Given a dictionary for B, conditions (i)-(iii) of Proposition 5.1 can be checked in $O(md)$ time.

Proposition 5.2: *The lex-min basis B of each basic feasible solution x is lex-positive.*

Proof: We must show the rows indexed $d+1, \ldots, m$ of (4.1) are lex-positive. Since x is feasible, we need only check rows with $\beta_i = 0$. Let $r \in B$ be a basic index corresponding to such a row. Suppose the smallest index j for which $\alpha_{i,j} \neq 0$ is such that $\alpha_{i,j} < 0$. Now if $r \leq m$ from Proposition 3.3 we have $\alpha_{i,r} = 1$, so $j < r$. On the other hand, if $r \geq m+1$, then trivially $j \leq m < r$. Also $j \in N$ since it indexes a column of (4.1) that is not a unit vector. Therefore $\bar{B} = B - r + j$ is a feasible degenerate pivot yielding a lexicographically smaller basis for the same feasible solution x. Since this is a contradiction, it must be that $\alpha_{i,j} > 0$ and so row i is lex-positive. □

Note that the proof of the proposition implies that in a lex-min basis B, if $i \in B$ and $i \geq m+1$, then necessarily $x_i > 0$.

Next we consider extreme rays. Here the situation is more complicated, due to the existence of parallel rays incident to distinct vertices, known as geometric rays, as illustrated in Section 2. In a non-simple polyhedron, the same geometric ray may appear in many dictionaries, ie. it may have many cobasis representations with the same basic solution. It is not necessarily true that a geometric ray incident with a vertex y will appear in the lex-min dictionary for this vertex. To see this, note that a dictionary can only identify at most d distinct rays, but a cone, for example, may have many more. There is, however, a way to identify the lex-min basis for any given geometric ray.

Proposition 5.3: *The lexicographically minimum basis representing the geometric ray $y + tz$ is lex-positive.*

Proof: Let B be any basis representing the geometric ray $y + tz$, with dictionary given by (3.4). Let $s \in N$ index the column in this dictionary corresponding to z, and let $a = A_B^{-1} A_s$. We consider an *augmented* dictionary by adding a new variable x_q, $q = m + d + 1$, and appending to (3.4) the new equation:

$$x_s + x_q = 1.$$

This corresponds to adding the constraint $x_s \leq 1$ to the original problem. The augmented dictionary has basis $B + q$ and basic feasible solution $(x_B, x_q) = (A_B^{-1} b, 1)$. Column s no longer defines a ray in the augmented dictionary, and it is possible to pivot on this column with variable x_q leaving the basis, and x_s entering with value one. After the pivot the augmented system has basis $B + s$, basic feasible solution $(x_B, x_s) = (A_B^{-1} b - a, 1)$, and represents the *vertex* $y + z$ of the augmented problem. By Proposition 5.2, the lex-min basis $\bar{B}_{\min}$ for this vertex is lex-positive. Let $\bar{N}_{\min}$ be the corresponding cobasis. Since the basic feasible solution has $x_s = 1$, $s \in \bar{B}_{\min}$. Let $B_{\min} = \bar{B}_{\min} - s$. We have $q \in \bar{N}_{\min}$, since if not x_q would be in the basis $\bar{B}_{\min}$ at value zero. As $q = m + d + 1$, it is the largest index, any degenerate pivot with q leaving the basis would produce a lexicographically smaller basis with the same basic feasible solution, contradicting the minimality of $\bar{B}_{\min}$. Since $q \in \bar{N}_{\min}$, the row in the dictionary for x_s is all zeroes except for a one in the columns for x_s and x_q.

We consider the pivot interchanging x_s and x_q in the augmented dictionary with basis $\bar{B}_{\min}$. Before the pivot, the column for x_q is $(a, 1)$, since increasing x_q to one changes the basic feasible solution from $(x_{B_{\min}}, x_q) = (A_B^{-1} b - a, 1)$ to $(x_{B_{\min}}, x_s) = (A_B^{-1} b, 1)$. The pivot preserves lex-positivity, since in the column for x_q only the last entry, corresponding to x_s, is positive. After the pivot the column for x_s is $(a, 1)$. If we delete the extra row from the augmented dictionary after the pivot, we get a dictionary for the original problem with lex-positive basis $B_{\min}$. It has basic feasible solution $x_{B_{\min}} = A_B^{-1} b$ and column s has value $-a$, so it represents the geometric ray $y + tz$. We claim that B is lexicographically at least as large as $B_{\min}$. This follows from the fact that $B + s$ is a basis for the vertex $y + z$ in the augmented problem, and so is lexicographically greater than or equal to the lex-min basis for this vertex, $\bar{B}_{\min} = B_{\min} + s$. □

A basis representing a geometric ray can be tested for lex-minimality quite efficiently. The next proposition is analogous to Proposition 5.1, Note conditions (i)-(iii) are identical.

Proposition 5.4: *B is the lex-min basis representing a geometric ray $y + tz$ with basic feasible solution x if and only if there does not exist $r \in B$ and $s \in N$ such that (i) $r > s$, (ii) $x_r = 0$, (iii) $(A_B^{-1} A_s)_r \neq 0$, and (iv) $(A_B^{-1} A_u)_r = 0$, where $A_B^{-1} A_u$ represents the ray z.*

Proof: The proof is similar to Proposition 5.1. If (i)-(iv) hold then the pivot interchanging r and s produces a smaller basis with the same properties. Conversely let $B_{\min}$ be the lex-min basis and B any other basis representing $y + tz$. Choose r and s as in Proposition 5.1. As shown there, (i)-(iii) must hold. It remains to observe that condition (iv) must hold or else the pivot would change column u by other than multiplication by a positive scalar, and the new basis would not represent $y + tz$. □

The conditions of Proposition 5.4 can be tested in $O(md)$ time given the current dictionary. The results of this section are summarized in the following theorem.

Theorem 5.5: *Each vertex y and each geometric ray $y + tz$ of P can be represented uniquely by its lex-min basis, which is lex-positive.* □

The goal of a vertex enumeration algorithm is to produce a minimum V-representation (2.2) for P. In this minimum representation each direction z producing one or more extreme rays should be output once. When the polyhedron P is a pointed cone, the result of Theorem 5.5 is enough to achieve this. Since there is only one vertex, the origin, the only geometric ray with direction z is tz. In this case the set of lex-min bases gives a minimum V-description of P. An important application of this is to facet enumeration of polytopes, see Section 6. For unbounded polyhedra with more than one vertex, we do not know any local necessary and sufficient condition to determine the lex-min basis for a ray (as opposed to a geometric ray). The following necessary condition allows some parallel geometric rays to be eliminated. Note conditions (i) and (iv) are the same as in Proposition 5.4.

Proposition 5.6: *B is the lex-min basis whose dictionary represents a ray z in column $A_B^{-1} A_u$ only if there does not exist $r \in B$ and $s \in N$ such that (i) $r > s$, (ii) $(A_B^{-1} A_s)_r > 0$, (iii) r is an index which minimizes the ratio $(A_B^{-1} b)_i / (A_B^{-1} A_u)_i$, over all $i \in B$ for which the denominator is positive, and (iv) $(A_B^{-1} A_u)_r = 0$.*

Proof: Conditions (i) - (iii) of the proposition imply that it is possible to make a feasible pivot to the basis $\bar{B} = B - r + s$, which is lexicographically smaller than B. Condition (iv) implies that the pivot does not change the column of the dictionary representing z. Let $\bar{x}$ be the basic feasible solution for basis $\bar{B}$. $\bar{B}$ is not necessarily lex-positive, but its dictionary represents the geometric ray $\bar{x} + tz$. By Proposition 5.3, the lex-min basis for this geometric ray is lex-positive. Since it is lexicographically no larger than $\bar{B}$, this proves B is not the lex-min basis of the ray z. □

Testing the conditions of Proposition 5.6 requires $O(md)$ time, but is more expensive than testing those of Propositions 5.1 and 5.4 due to the additional ratio test required for condition (iii).

6. Reverse Search

Based on Theorems 4.4 and 5.5 we can design pivoting algorithms to generate the vertices and extreme rays of P. An algorithm of this type is initiated with the lex-positive basis B^* and uses lexicographic pivoting to generate all lex-positive bases. A duplicate free list of vertices and geometric rays is obtained by outputting only the lex-min bases for each vertex and geometric ray. The algorithm can be described as a search of the graph whose nodes are lex-positive bases, and whose edges correspond to lexicographic pivots between these bases. A standard graph traversal algorithm, such as depth first search, could be used but suffers from the disadvantage that it is necessary to keep a list of all bases discovered. Even for rather small inputs, the size of such a list can be prohibitively large. *lrs* implements reverse search (see [2]), which is a method that enables the graph to be searched without maintaining a list of visited nodes. In the reverse search algorithm, only the current basis B, cobasis N and corresponding dictionary are stored.

The research search algorithm makes use of the following four functions.

(a) *pivot*(B, r, s). B is the basis of the current dictionary, $r = B_i$ is a basic index and $s = N_j$ is a cobasic index. The dictionary is pivoted to the new basis $B - r + s$ as described in Section 3. The basic and cobasic indices B and N are updated by setting $B_i = s$ and $N_j = r$.

(b) *selectpivot*(B, r, j). B is the basis of the current dictionary, which is assumed non-optimal. *selectpivot* returns indices $r \in B$ and $s \in N$ chosen by the lexicographic pivot rule applied to the dictionary with basis B. First the least index $s = N_j$ is found with negative coefficient in row zero, ie. such than $(A_B^{-1} A_s)_0 < 0$. Then the lexicographic ratio test is performed on this column as described in Section 4, ie. we compute $r = lexminratio(B, s)$. The indices r and j are returned.

(c) *lexmin*(B, s). B is the current basis representing a vertex y and s is either zero, or else $s \in N$ and the corresponding column of the dictionary for B represents a ray. When s is zero, *lexmin* determines if B is the lex-min basis for y, and if so returns *true*. Otherwise $s \in N$ and this column of the dictionary represents a ray z. *lexmin* determines if B is the lex-min basis for the geometric ray $y + tz$, and if so returns *true*. The operations required are given in Propositions 5.1 and 5.4.

(d) *reverse*(B, u, v). Given basis B and cobasic index $v \in N$, *reverse* determines if there is a basic index $u \in B$ such that the lexicographic pivot rule applied to the dictionary with basis $\bar{B} = B - u + v$ generates a pivot back to B. In other words, *reverse* determines if there is an index $u \in B$ so that *selectpivot* applied to the dictionary with basis $B - u + v$ would compute the indices $r = v$ and $s = u$. This will happen if the first negative coefficient in row zero of the dictionary for $\bar{B}$ has index u, and if $v = lexminratio(\bar{B}, u)$. In this case, *reverse* returns *true* with the index u, otherwise it will return *false*. If the column v represents a ray, the ray is output if *lexmin*(B, v) is *true*. *reverse* can be implemented by using *pivot* and *selectpivot*, but this is very inefficient. An efficient implementation is discussed below.

Using these four functions, the reverse search algorithm *lrs* can be described by the pseudo-code given in Figure 6.1. It is assumed that the dictionary with current basis B is available to all functions.

A general discussion of reverse search, proof of correctness and complexity analysis is given in[5], to which the reader is referred for more information. The main **while** loop is executed for each basis B, starting with the optimum basis. Each cobasic column of the current dictionary is examined by *reverse* to see if there is a lex-positive pivot using this column, for which the resulting dictionary pivots back to B using the lexicographic simplex method. If so the pivot is formed, and the **while** loop is executed for each column of the new dictionary. When the **while** loop terminates for a given basis, *selectpivot* and *pivot* are used to return to its parent in the reverse search tree, and the **while** loop is continued for this basis. Note that for this it is essential that the value of j be correctly restored, as is done in *selectpivot*. The **until** statement is used to terminate the algorithm after the last column of the starting optimum dictionary has been examined.

Inspection of the pseudo-code shows that there are two pivots for each basis except the initial basis B^*: one in the **while** loop when a new basis is found, and one in the backtracking step, when a simplex pivot is used to move to the parent basis. Since pivoting is the most time consuming operation, a speedup can be achieved by saving the recent dictionaries in a cache, so they can be reloaded rather than recomputed. More critically, it is clear that *reverse* is executed d times for each basis and so must be implemented efficiently. This can be achieved by using the following proposition.

```
B = B*; j = 1;
repeat
    while j ≤ d
        { v = N_j;
        if reverse(B, u, v)
        then { pivot(B, u, v);            // new basis found //
               if lexmin(B, 0) then output current vertex;
               j = 1;
             }
          else  j = j + 1;
        }
    selectpivot(B, r, j);                 // backtrack //
    pivot(B, r, N_j);
     j = j + 1;
until j > d and B = B*.
```

Figure 6.1: Pseudo-code for *lrs*

Proposition 6.1: *Given basis B, index $v \in N$, let $a = A_B^{-1} A_v$, and for any $t = 1, \ldots, m$ let w^t be the vector of coefficients of row t of the current dictionary (3.4). The function $reverse(B, u, v)$ is true and returns $u = B_i$ if and only if (i) $w_v^0 < 0$, (ii) $u = B_i = lexminratio(B, v) \neq 0$, and (iii) setting $\bar{w} = w^0 - a_0 w^i / a_i$, we have $\bar{w}_j \geq 0$, for all $j \in N$, $j < u$.*

Proof: Let $\bar{B} = B - u + v$. First suppose $reverse(B, u, v)$ is *true*, which implies that *selectpivot* applied to the dictionary with basis $\bar{B}$ returns $r = v$ and $s = u$. This implies that $v = lexminratio(\bar{B}, u) \neq 0$. By Proposition 4.1(b), $u = lexminratio(B, v)$, giving (ii). It also implies that $a_i > 0$, and so $\bar{w}$ is well defined. It follows from the pivot operation (Section 3) that $\bar{w}$ is the vector of coefficients of row zero of the dictionary with basis $\bar{B}$. Since *selectpivot* chooses $s = u$, $\bar{w}_u$ must be the first negative component of $\bar{w}$, so (iii) holds. Also by the pivot formula, $\bar{w}_u = w_v^0 / a_i$. Since a_i is positive, (i) holds.

For the converse, assume (i) - (iii) hold. Define $\bar{B}$ as above. As argued, $\bar{w}$ is the vector of coefficients of row zero of the dictionary corresponding to $\bar{B}$. From (ii) $a_i > 0$ and from (i) $w_v^0 < 0$, so $\bar{w}_u < 0$. Together with (iii) this implies that *selectpivot* applied to the dictionary with basis $\bar{B}$ will select u as the entering index. Since $u \neq 0$, we can apply Proposition 4.1(b), getting $v = lexminratio(\bar{B}, u)$. Therefore v is chosen as the leaving index by *selectpivot* and so $reverse(B, u, v)$ is true. □

From Proposition 6.1 we see that *reverse* requires a lexicographic ratio test for each negative coefficient in row zero of the dictionary. If the ratio test succeeds, meaning the column does not represent an extreme ray, part of the row zero of the updated dictionary is computed. The ratio test dominates the cost of this, requiring $O(md)$ time for degenerate dictionaries in the worst case. For non-degenerate dictionaries, the ratio test requires only $O(m)$ time, which since $m > d$ is also the time for testing all of the conditions (i)-(iii).

Referring to the pseudo-code in Figure 6.1, we see that the total time required by *reverse* for a basis B is $O(md^2)$ for degenerate dictionaries and $O(md)$ for non-degenerate dictionaries. The time required for an execution of *pivot*, *lexmin* or *selectpivot* is $O(md)$. This proves the following result.

Theorem 6.2: *lrs finds all bases and hence all vertices of a polyhedron in time $O(md^2)$ per basis and $O(md)$ space. It finds all vertices of a simple polyhedron in time $O(md)$ per vertex.* □

7. Implementation Issues and Integer Pivoting

To minimize space, *lrs* uses the reduced form of the dictionary (3.4) given by

$$x_B = A_B^{-1} b - A_B^{-1} A_N x_N \tag{7.1}$$

which is stored as a $(m+1) \times (d+1)$ array. Note that the signs of the coefficients of cobasic variables are the reverse of those in (3.4). Therefore the optimum dictionary is characterized by a row zero having non-positive coefficients. For any non-optimum dictionary, the candidates for entering variable for the simplex method are those with a positive coefficient in row zero. For definiteness, we choose the one with minimum index. For the lexicographic ratio test required to find the leaving variable, the rows of (4.1) required for (4.2) are recovered column by column, using Proposition 3.1, until a unique minimum is found. We can interpret lex-positivity in the context of the dictionary (3.4). B is a lex-positive basis if and only if for each $i \in B$ with $i \geq m+1$ and basic feasible solution $x_i = 0$, the first non-zero coefficient in its corresponding row of (7.1) is negative.

The basic and cobasic indices B and N are maintained in increasing order in *lrs*. In order to avoid moving the data in the dictionary, pointers are maintained to the actual row and column locations in the dictionary. This ordering allows certain operations to be optimized. For example, the test $B = B^*$ can be achieved by checking if $B_m = m$. Similarly the tests required in *select-pivot*, *lexmin* and *reverse* can be speeded up by processing the indices in order.

The reverse search method is extremely sensitive to numerical error. A single mistake in the sign of a dictionary element can mean that an entire subtree of the reverse search tree is not discovered. For that reason, exact arithmetic is used. *lrs* uses arrays to hold long integers, using the data format and basic routines given by Gonnet and Baeza-Yates [16]. The division routine is based on Knuth [19] and was implemented by Jerry Quinn.

The earlier version *rs* used rational arithmetic, and each entry in the dictionary was stored as a rational in reduced form, requiring *gcd* computations after each arithmetic operation. Most of these computations are eliminated in *lrs* by the use of integer pivoting method of Edmonds [15] (which is connected to Cramer's rule, see the appendix of Chvátal [13].) In integer pivoting, only the numerators of coefficients of the dictionary (7.1) are stored, with respect to a common denominator, which is the absolute value of the determinant of the current basis. The absolute value of the determinant is used so that the signs of the numerators agree with the signs of the rational numbers they represent. Let $a_{i,j}$, $0 \leq i \leq m$, $0 \leq j \leq d$ denote the array of coefficients of (7.1), and let $det(B)$ be the determinant of the current basis B. If a pivot is to be performed on row r and column s, the updated (barred) coefficients are given by:

$$\bar{a}_{i,j} = (a_{i,j} a_{r,s} - a_{i,s} a_{r,j}) / det(B)$$

$$\bar{a}_{r,j} = -a_{r,j}, \quad \bar{a}_{i,s} = a_{i,s}, \quad \bar{a}_{r,s} = \det(B), \quad det(\bar{B}) = a_{r,s}$$

where in the above formulae, $i \neq r$ and $j \neq s$. It can be shown that the integer division has no remainder. Since no *gcd* computations are required in integer pivoting, the only *gcd* operations performed in *lrs* are before printing the output. Empirically, integer pivoting appears to be between two and ten times faster than rational pivoting, with *gcd* computation using Euclid's algorithm.

8. Other Functions of *lrs*

In this section we briefly described other functions of *lrs* that are built on top of the basic function of vertex enumeration. The details are given elsewhere, as noted.

8.1. Facet Enumeration

The facet enumeration problem is to produce an H-representation (2.1) of P from a V-representation (2.2). In *lrs* a standard lifting technique (see for example Ziegler [20]) is used to convert the facet enumeration problem to an equivalent vertex enumeration problem. The input V-representation is lifted to a pointed cone in one higher dimension, for which the two problems are equivalent. Specifically, each vertex $(a_1, \ldots, a_d)$ of P is transformed to the inequality

$$x_1 + a_1 x_2 + \cdots + a_d x_{d+1} \geq 0 \tag{8.1}$$

and each ray $(a_1, \ldots, a_d)$ of P is transformed to the inequality

$$a_1 x_2 + \cdots + a_d x_{d+1} \geq 0.$$

The resulting system of inequalities describes a pointed cone $\bar{P}$ in $d+1$-dimensions. A ray $(z_1, \ldots, z_{d+1})$ of $\bar{P}$ corresponds to the facet

$$z_1 + z_2 x_1 + \cdots + z_{d+1} x_d \geq 0$$

of P. Note that no lifting is required if the input polyhedron P is a pointed cone. Also, if the input is the V-representation of a polytope containing the origin, it can also be solved without lifting by interpreting the input points as inequalities (8.1), with x_1 set to the constant one. Somewhat remarkably, although lifting increases degeneracy, *lrs* sometimes runs faster on lifted polytopes. This is due to the fact that rays are detected more efficiently than vertices, each of which require one (or two) additional pivots.

8.2. Voronoi Vertices

Given a set of m points in R^d, it is required to find the set of Voronoi vertices, each characterized by being the centre of an empty hypersphere spanned by at least $d+1$ input points. It is well known (see for example, Edelsbrunner [14]) that the Voronoi vertices of a set of points in R^d can be obtained by solving a vertex enumeration problem in R^{d+1}. Each input point $(a_1, \ldots, a_d)$ is transformed to the inequality

$$(a_1^2 + \cdots + a_d^2) - 2a_1 x_1 - \cdots - 2a_d x_d + x_{d+1} \geq 0$$

The resulting system of inequalities describes a polyhedron $\bar{P}$ in R^{d+1}. There is a one-to-one correspondence between the vertices of $\bar{P}$ and the Voronoi vertices of the input set of data. Indeed, each vertex $(y_1, \cdots, y_{d+1})$ of $\bar{P}$ projects to the Voronoi vertex $(y_1, \cdots, y_d)$. The inequalities described by the cobasic indices for a vertex of $\bar{P}$ also correspond to the input points defining the corresponding Voronoi vertex.

8.3. Volume Computation

Given a set of m points in R^d it is required to find the volume of their convex hull. Let P be the polytope spanned by the input data points. As described in subsection 8.1, the facet enumeration of P can be obtained from the rays of the $d+1$-dimensional cone $\bar{P}$. It can be shown that the lex-positive bases of $\bar{P}$ form a decomposition of the polytope P into non-overlapping simplices. This follows from the fact that the lex-positive bases of $\bar{P}$ correspond to the vertices of a simple polyhedron obtained by a suitable perturbation of $\bar{P}$. The dual of this simple polyhedron is simplicial, and its projection gives the required decomposition of P. As we saw in Section 7, the determinant of each basis of $\bar{P}$ is readily available during the computation. The determinant is a equal to $d!$ times the volume of the corresponding simplex of P. Hence summing these determinants and dividing by $d!$ gives the required volume of P. See Büeler et al. [11] for more details and a comparison with other volume computation methods.

8.4. Estimation

The number of vertices in the V-representation of a polyhedron represented by m inequalities in R^d can vary from one to $m^{\lfloor d/2 \rfloor}$. P may have as many as m^d feasible bases, of which as many as $m^{\lfloor d/2 \rfloor}$ may be lex-positive. Clearly, for even small values of m and d, if P achieves these upper bounds the vertex enumeration problem is intractable. As the running time of *lrs* is directly proportional to the number of bases computed, it is useful to estimate this number. Reverse search is amenable to a technique of tree estimation due to Hall and Knuth [17]. As described by Avis and Devroye [3], *lrs* can be used to estimate the number of lex-positive bases, and also the number of vertices and rays (or facets for facet enumeration problems). Similarly estimates for the volume of a polytope can be obtained. The estimates are unbiased, and techniques are given to lower the variance. In spite of the enormous range of the quantities to be estimated, the estimates obtained appear to give a good indication of the tractability of solving the given problem completely.

8.5. Restart Capability

A feature of algorithms based on reverse search is that they can easily be interrupted and restarted in the middle of a computation. In the vertex enumeration setting, it is necessary to record only the indices of the current cobasis before interrupting the program. Then using this cobasis and the original input file, the current dictionary can be recomputed, and the computation resumed from this node of the reverse search tree. If cacheing is used, the cache is lost after the program is interrupted, but is restored automatically as new pivots are made after restarting.

9. Computational Results

In [4] some preliminary computational experience was given on a set of seven test problems. To illustrate the evolution of the code, we ran both the original program *ve*01 and version 2.3 of *lrs* on these test problems, using the latest incarnation of *mutt*, a DEC AlphaServer 1000 4/23. The results are shown in Figure 9.1. In the table, #V, #R, #B refer respectively to the number of vertices, rays and bases computed. Note that due to lexicographic pivoting, *lrs* computes considerably fewer bases on the degenerate problems 2, 3 and 6. Problem 1 has a matrix generated uniformly in the range -1000..1000 and b vector all ones. Problem 2 is due to Akihisa Tamura, and has all data from the set {0,-1,1}. Problem 3 is the truncated Metric Cone on four points, consisting essentially of all triangle inequalities and non-negativity constraints on these four points. Problems 4 and 5 were constructed arbitrarily with integer data in the range -100..100. Problem 6 was arbitrarily constructed with matrix entries in the range -10..10 and b vector 1..13. Problem 7 is a Kuhn-Quandt problem, with matrix entries randomly chosen in the range 0..1000 and b vector entries all 10000. Its solution required integers with up to 63 decimal digits.

					ve01		lrs	
Problem	m	d	#V	#R	#B	secs	#B	secs
1	34	4	31		31	2.81	31	.06
2	16	5	18		1247	7.48	76	.05
3	19	6	8		10845	86.43	188	.12
4	12	7	54		54	.43	54	.05
5	14	9	89	33	97	1.14	94	.09
6	23	10	332	302	3656	83.65	824	1.69
7	20	10	1188		1188	208.98	1188	2.78

Figure 9.1: Computational Results

One of the largest problems solved was a configuration polytope with 96 inequalities in 86 dimensions. The output of 323,188 vertices was obtained by computing 1,621,760 bases in a computation lasting 30 days. Even larger problems have been solved using a parallel version of *lrs* described by Brüngger et al. in [10]. For example, they report computing the more than 3 million vertices and 57 million bases of a configuration polytope defined by 71 inequalities in 60 dimensions. More extensive computational experience, and an empirical analyses of the various speedups, is reported in Avis[7].

10. Acknowledgements

This paper was written while the author was on leave at IASI-CNR in Rome, whose support is gratefully acknowledged. Various people helped with aspects of the development of *lrs*, including David Bremner, Komei Fukuda, Ambros Marzetta, and Jerry Quinn. Finally, I would like to thank the users of *lrs* for suggestions, encouragement....and reporting bugs!

References

1. *lrs Home Page*, November 1997. ftp://mutt.cs.mcgill.ca /pub/C/lrs.html
2. D. Avis and K. Fukuda, ‘‘A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra,’’ *Discrete and Computational Geometry*, vol. 8, pp. 295-313, 1992.
3. D. Avis and L. Devroye, ‘‘Estimating the Number of Vertices of a Polyhedron,’’ in *Snapshots of Computational and Discrete Geometry*, ed. D. Avis and P. Bose, vol. 3, pp. 179-190, School of Computer Science, McGill University, 1994. ftp://mutt.cs.mcgill.ca/pub/doc/avis/AD94a.ps.gz
4. D. Avis, ‘‘A C Implementation of the Reverse Search Vertex Enumeration Algorithm,’’ in *RIMS Kokyuroku 872*, ed. H. Imai, Kyoto University, May 1994. ftp://mutt.cs.mcgill.ca /pub/doc/avis/Av94a.ps.gz
5. D. Avis and K. Fukuda, ‘‘Reverse Search for Enumeration,’’ *Discrete Applied Math*, vol. 6, pp. 21-46, 1996.
6. D. Avis, D. Bremner, and R. Seidel, ‘‘How Good are Convex Hull Algorithms?,’’ *Computational Geometry: Theory and Applications*, vol. 7, pp. 265-301, 1997.
7. D. Avis, ‘‘Computational Experience with the Reverse Search Vertex Enumeration Algorithm,’’ *Optimization Methods and Software*, 1998 (to appear). ftp://mutt.cs.mcgill.ca /pub/doc/avis/Av98b.ps.gz
8. R. G. Bland, ‘‘New Finite Pivoting Rules for the Simplex Method,’’ *Math. Operations Research*, vol. 2, pp. 103-107, 1977.
9. D. Bremner, K. Fukuda, and A. Marzetta, ‘‘Primal-Dual Methods of Vertex and Facet Enumeration,’’ *Discrete and Computational Geometry*, vol. 20, pp. 333-358, June 1997.
10. A. Brüngger, A. Marzetta, K. Fukuda, and J. Nievergelt, *The Parallel Search Bench ZRAM and its Applications*, 1997. ftp://ftp.ifor.math.ethz.ch/pub/fukuda/reports/zram_poc970924.ps.gz
11. B. Büeler, A. Enge, and K. Fukuda, *Exact Volume Computation for Polytopes: A Practical Study*, January 1998. ftp://ftp.ifor.math.ethz.ch/pub/fukuda/reports/vol_article980112.ps.gz
12. M. Bussieck and M. Luebbecke, ‘‘The Vertex Set of a 0/1-Polytope is Strongly P-Enumerable,’’ *ISMP ’97*, Lausanne, August 1997.
13. V. Chvátal, *Linear Programming,* W.H. Freeman, 1983.
14. H. Edelsbrunner, *Algorithms in Combinatorial Geometry,* Springer-Verlag, 1987.
15. J. Edmonds and J.-F. Maurras, ‘‘Note sur les Q-matrices d’Edmonds,’’ *Recherche Opérationelle (RAIRO)*, vol. 31, pp. 203-209, 1997.
16. G.H. Gonnet and R. Baeza-Yates, *Handbook of Algorithms and Data Structures-2nd Edition,* Addison-Wesley, 1991.
17. M. Hall and D.E. Knuth, ‘‘Combinatorial Analysis and Computers,’’ *Am. Math. Monthly*, vol. 72, pp. 21-28, 1965.
18. J. Ignizio and T. Cavalier, *Linear Programming,* Prentice Hall, 1994.
19. D. E. Knuth, *The Art of Computer Programming, Vol 2: Seminumerical Algorithms,* Addison-Wesley, Reading, MA, 1981.
20. G. Ziegler, *Lectures on Polytopes,* Springer, 1994, revised 1998. Graduate Texts in Mathematics.