A C Implementation of the Reverse Search Vertex Enumeration Algorithm

David Avis

School of Computer Science
McGill University
3480 University, Montreal, Quebec H3A 2A7

Revised June 1993

ABSTRACT

This report documents a C implementation of the reverse search vertex enumeration algorithm for convex polyhedra of Avis and Fukuda. The implementation uses multiple precision rational arithmetic and contains a few improvements over the original algorithm, resulting in a small reduction in the complexity in most cases. For a polyhedron with n_0 inequalities in d non-negative variables and non-degenerate origin, it finds all bases in time $O(n_0 d^2)$ per basis. It is also shown how the implementation can be improved to run in time $O(n_0 d \; min(n_0, d))$ time per basis for any polyhedron. This implementation can handle problems of quite large size, especially for simple polyhedra (where each basis corresponds to a vertex). Computational experience is included in the report.

1. Background

A new enumeration method called "Reverse Search" was recently introduced by Avis and Fukuda[1] (hereafter referred to as AF) and applied to the problem of finding all vertices of a convex polyhedron. This report documents a C implementation of the algorithm. The implementation uses multiple precision rational arithmetic and contains a few improvements over the original algorithm. It can handle problems of quite large size, especially for simple polyhedra. Computational experience is included in the report.

This implementation in C was motivated by an earlier implementation by K. Fukuda and I. Mizukoshi in *Mathematica*[5]. Their implementation demonstrates the simplicity and effectiveness of the method, and provides impressive graphical output (see, eg. Figure 3.5). However, due to the inherent limitations of *Mathematica*, it is only computationally feasible for small problems. The C implementation described here runs several hundred times faster for the medium sized problems described in Section 3. Most of Section 1 can be omitted by potential users, who may turn directly to Section 2. If the problem is not in the standard form of (1.3), it will be necessary to refer to Section 1.2.

It is assumed throughout that the reader is familiar with the paper AF. Briefly and informally, the algorithm works as follows. Suppose we have a system of linear inequalities defining a polyhedron in R^d and a vertex of that polyhedron. A vertex is specified by giving the indices of d inequalities whose bounding hyperplanes intersect at the vertex. For any given linear objective function, the simplex method generates a path along edges of the polyhedron until a vertex maximizing this objective function is found. For simplicity, let us assume for the moment that the optimum vertex is contained on exactly d bounding hyperplanes. The path is found by pivoting, which involves interchanging one of the equations defining the vertex with one not currently used. The path chosen from an initial given vertex depends on the pivot rule used. In fact, care must be taken because some pivot rules generate cycles and do not lead to the optimum vertex. However, a particularly simple rule, known as Bland's rule or the least subscript rule[2], guarantees a unique path from any starting vertex to the optimum vertex. If we look at the set of all such paths from all vertices of the polyhedron, we get a spanning tree of the edge graph of the polyhedron rooted at the optimum vertex. Our algorithm simply starts at an "optimum vertex" and traces out the tree in depth first order by "reversing" Bland's rule.

A remarkable feature is that no additional storage is needed at intermediate nodes in the tree. Going down the tree we explore all valid "reverse" pivots in lexicographical order from any given intermediate node. Going back up the tree, we simply use Bland's 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 lexicographic "reverse" pivot, etc. The algorithm is therefore non-recursive and requires no stack or other data structure. One possible difficulty arises at so-called degenerate vertices, vertices which lie on more than d bounding hyperplanes. It is desirable to report each vertex once only, and this can be achieved without storing the output and searching. By using duality, we can also use this algorithm for enumerating the facets of the convex hull of a set of points in R^d. It can also be used for enumerating all of the vertices of the Voronoi Diagram of a set of points in R^d, since this can be reformulated as a convex hull problem in R^{d+1} (see [4]). We now give a formal statement of the problem. The labelling of the variables is slightly different than AF in order to conform with C indexing conventions. For more information and background on Linear Programming and vertex enumeration of polyhedra, the reader is referred to [3].

1.1. Dictionaries

Let A be a $m \times n$ matrix, with columns indexed by the set $E = \{0, 1, 2, \ldots, n-1\}$. Fix distinct indices f and g of E. Consider the system of equations:

$$A\ x = 0, \quad x_g = 1. \tag{1.1}$$

For any $J \subseteq E$, x_J denotes the subvector of x indexed by J, and A_J denotes the submatrix of A consisting of columns indexed by J. A *basis* B for (1.1) is a subset of E of cardinality m containing f but not g, for which A_B is nonsingular. We will only be concerned with systems (1.1) that have at least one basis, and will assume this for the rest of the paper. Given any basis B, we can transform (1.1) into the *dictionary*:

$$x_B = -A_B^{-1} A_N\ x_N = \bar{A}\ x_N, \tag{1.2}$$

where $N = E - B$ is the $co-basis$, and $\bar{A}$ denotes $-A_B^{-1} A_N$. $\bar{A}$ is called the *coefficient matrix* of the dictionary, with rows indexed by B and columns indexed by N, so that $\bar{A} = (\bar{a}_{ij} : i \in B,\ j \in N)$. Note that the co-basis always contains the index g.

A variable x_i is *primal feasible* if $i \in B - f$ and $\bar{a}_{ig} \geq 0$. A variable x_j is *dual feasible* if $j \in N - g$ and $\bar{a}_{fj} \leq 0$. A dictionary is *primal feasible* if x_i is primal feasible for all $i \in B - f$ and *dual feasible* if x_j is dual feasible for all $j \in N - g$. A dictionary is *optimal* if it is both primal and dual feasible. An optimal dictionary is shown schematically in Figure 1.1. A *basic solution* to (2.1) is obtained from a dictionary by setting $x_{N-g} = 0$, $x_g = 1$. If any basic variable has value zero, we call the basic solution and corresponding dictionary *degenerate*.

Degenerate optimal dictionaries pose a problem for reverse search, since there may be many optimal dictionaries, and each has to be enumerated. For this reason we define an *augmented dictionary* which always has a unique "optimum" solution. The augmented dictionary is just the original optimal dictionary if this is non-degenerate. Otherwise, let $x_B = \bar{A} x_N$ be a degenerate optimal dictionary. Let $B' \subseteq B$ denote the indices of the variables with value zero in the corresponding basic solution and the index f. We augment $\bar{A}$ by a column with index $g' = n + 1$, defined by:

$$\bar{a}_{ig'} = \begin{cases} 1 & \bar{a}_{ig} = 0 \\ \bar{a}_{ig} & otherwise. \end{cases}$$

$$
\begin{array}{c|c|c|}
 & N-g & g \\
\cline{2-3}
f & \ominus\ \ominus\ \ominus\ \ominus\ \ominus\ \ominus & \\
\cline{2-3}
B-f & & \begin{array}{c}\oplus\\ \oplus\\ \oplus\\ \oplus\\ \oplus\end{array} \\
\cline{2-3}
\end{array}
\quad = \ \bar{A}
$$

Figure 1.1 : An Optimal Dictionary

($\oplus$ = non-negative entry, $\ominus$ = non-positive entry)

Let $N' = N - g + g'$. This augmented dictionary is shown schematically in Figure 1.2.

$$
\begin{array}{c|c|c|c|}
 & N'-g' & g & g' \\
\cline{2-4}
f & -\ -\ -\ -\ -\ - & & \\
\cline{2-4}
B'-f & & \begin{array}{c}0\\0\\0\end{array} & \begin{array}{c}+\\+\\+\end{array} \\
\cline{2-4}
B-B'-f & & \begin{array}{c}+\\+\\+\end{array} & \begin{array}{c}+\\+\\+\end{array} \\
\cline{2-4}
\end{array}
\quad = \ \bar{A}
$$

Figure 1.2 : An Augmented Degenerate Optimal Dictionary

1.2. Vertex enumeration for Polyhedra

A (convex) polyhedron P is the solution set to a system of n_0 inequalities in d non-negative variables:

$$P = \{ y \in R^d \mid A'y \le b,\ y \ge 0 \}, \tag{1.3}$$

where A' is an $n_0 \times d$ matrix and b is a n_0-vector. A *vertex* of the polyhedron is a vector $y \in P$ that satisfies a linearly independent set of d of the inequalities as equations. The *vertex enumeration problem* for P is to enumerate all of its vertices. Polyhedra not in the form (1.3) can be transformed into this form once a vertex is found, see AF for details.

Let $m = d + 1$, $n = n_0 + d + 2$, $f = 0$, $g = n - 1$, $B = \{0, n - m, n - m + 1, \ldots, n - 2\}$ and $N = \{1, \ldots, n - m - 1, n - 1\}$. Form the dictionary $\bar{A}$ from A' and b as follows:

(a) all signs of A' are reversed;
(b) vector b is appended as an additional column;
(c) a row zero is appended with all entries -1;
(d) if any component of b is zero, form the augmented dictionary as described in Section 1.1.

The program performs a reverse search on the optimum augmented dictionary $x_B = \bar{A} x_N$.

Polyhedra not in the form (1.3) can be transformed into this form once a vertex is found. These standard results are given here for completeness.

1.2.1. Polyhedra Without Non-Negativity Constraints

In this section we show how to convert a polyhedron without non-negativity constraints into a polyhedron in the form (1.3). Consider the polyhedron

$$\bar{P} = \{x \in R^d \mid \bar{A}x \le \bar{b}\},$$

defined by an $(n_0 + d) \times d$ matrix $\bar{A}$. If $\bar{P}$ has vertices, then $n_0 \ge 0$ and $\bar{A}$ must have full column rank. In order to use the vertex enumeration algorithm, we transform $\bar{P}$ in to a polyhedron of form (1.3). To do this it is necessary to have some vertex $\bar{x}$ of $\bar{P}$. Such a vertex can always be found by solving a linear program with constraint set $\bar{P}$. The transformation consists essentially of mapping the known vertex $\bar{x}$ to the origin, and of mapping d hyperplanes defining $\bar{x}$ onto the d coordinate planes.

Let A_1 be a d dimensional submatrix of of $\bar{A}$ formed by choosing d linearly independent rows of $\bar{A}$ whose constraints are satisfied as equations by $\bar{x}$. Let A_2 be the remainder of the matrix $\bar{A}$. Similarly partition $\bar{b}$ into b_1 and b_2. Then we have

$$A_1 \bar{x} = b_1$$

$$A_2 \bar{x} \le b_2. \tag{1.4}$$

Such a partition must always exist if $\bar{x}$ is a vertex of $\bar{P}$. Next define the d-dimensional vector y by

$$y = b_1 - A_1 x$$

Note that for any x in $\bar{P}$ we have $y \ge 0$. Since A_1 has full rank we can solve for x getting

$$x = A_1^{-1}(b_1 - y) \tag{1.5}$$

Substituting into (1.4) we have the system

$$-A_2\, A_1^{-1} y \le b_2 - A_2\, A_1^{-1} b_1$$

Therefore we get a polyhedron of form (1.3) by making the identification

$$A' = -\, A_2\, A_1^{-1}, \quad b = b_2 - A_2\, A_1^{-1} b_1$$

It is not hard to show that the vertices of (1.3) are in one-to-one correspondence with the vertices of $\bar{P}$. Vertices of (1.3) can be transformed into vertices of $\bar{P}$ by using the system of equations (1.5).

1.2.2. Facet Enumeration of the Convex Hull of a Set of Points

In this section we show how to transform the problem of enumerating all of the facets of the convex hull of a set of points into a vertex enumeration problem for a polyhedron. Let $P^* = \{p_1, p_2, \ldots, p_n\}$ be a set on n points in R^d. We assume that the point set is d-dimensional and, by translating the points if necessary, that the origin lies in the interior of the convex hull of P^*. An inequality $ay \le 1$ defines a facet of the convex hull of P^* if $ap_i \le 1,\ i = 1, \ldots, n$, and equality is obtained for an affinely independent subset of at least d of the points of P^*.

Form the $n \times d$ matrix $\bar{A}$ by letting row i be p_i. Then standard arguments show that the vertices of

$$\bar{P} = \{x \in R^d \mid \bar{A}x \le 1\}, \tag{1.6}$$

correspond to the facets of the convex hull of P^*. In particular, if $\bar{x}$ is a vertex of $\bar{P}$, then $\bar{x}y \le 1$ defines a facet of the convex hull of P^*. Using the results of the previous subsection, (1.6) can be transformed into the form of (1.3).

1.3. Pivoting

A *pivot* (r, s) on a basis B, and corresponding dictionary $x_B = \bar{A} x_B$, is an interchange of some $r \in B - f$ with some index $s \in N - g$ giving a new basis B'. The new coefficient matrix $A' = (a'_{ij})$ is given by

$$a'_{sr} = \frac{1}{\bar{a}_{rs}}, \quad a'_{ir} = \frac{\bar{a}_{is}}{\bar{a}_{rs}}, \quad a'_{sj} = -\frac{\bar{a}_{rj}}{\bar{a}_{rs}}, \quad a'_{ij} = \bar{a}_{ij} - \frac{\bar{a}_{is}\bar{a}_{rj}}{\bar{a}_{rs}}, \qquad (\, i \in B - r,\ j \in N - s\,). \quad (1.7)$$

A pivot is *primal feasible* if the new basic solution obtained from (1.2) is non-negative. Note that a pivot maintains the property that $f \in B$ and $g \in N$. A basic solution to (1.2) is *degenerate* if $x_i = 0$ for some $i \in B - f$. Let B be a basis such that the dictionary (2.2) is primal feasible. We call a pivot that is chosen by the following rule a *Bland* pivot.

Bland's Rule.

(1) Let s be the smallest index such that x_s is dual infeasible, that is, $\bar{a}_{fs} > 0$.

(2) Set $\lambda = \min\{ -\frac{\bar{a}_{ig}}{\bar{a}_{is}} : i \in B - f,\ \bar{a}_{is} < 0 \}$. Let r be the smallest index obtaining this minimum.

The pivot (r, s) maintains the primal feasibility of the dictionary. If step (1) does not apply, the dictionary is also dual feasible and hence optimal.

As remarked in the previous section, if the optimal dictionary is degenerate there may be multiple optimal dictionaries. However, in the augmented dictionary defined in the last section, all except one of the optimal dictionaries will contain negative entries in column g'. With respect to g' the other optimal dictionaries are "primal infeasible" but dual feasible. If we use a dual form of Bland's rule (letting column g' play the role of column g), we will pivot to the unique optimum augmented dictionary.

Dual Bland's Rule

(1) Let $r \in B' - f$ be the smallest index that is primal infeasible, that is $a_{rg'} < 0$.

(2) Set $\lambda = \min\{ -\frac{\bar{a}_{fj}}{\bar{a}_{rj}} : j \in N' - g',\ \bar{a}_{rj} > 0 \}$. Let r be the smallest index attaining this minimum.

The pivot (r, s) maintains the dual feasibility of the dictionary. If step (1) does not apply, the dictionary is optimal.

In AF, the problem of a degenerate optimal dictionary is overcome by a kind of two phase method. Phase 1 enumerates optimal dictionaries by reversing Dual Bland's rule starting with the unique optimum augmented dictionary. For each such dictionary, phase 2 enumerates all primal dictionaries leading to this dictionary by reversing Bland's rule.

In this implementation, we use essentially the same approach expressed a little differently. First we combine both pivot rules so that a repeated application of the Combined Rule leads to the unique optimum augmented dictionary. This Combined Pivot rule is then reversed to enumerate all primal feasible dictionaries for the original problem. This rule works always on the augmented dictionary defined in the last section.

Combined Pivot Rule

(1) If $\bar{A}$ is not dual feasible choose pivot (r, s) according to Bland's rule, otherwise

(2) Choose pivot (r, s) according to Dual Bland's rule.

If $\bar{A}$ is any primal feasible dictionary derived from the original optimum augmented dictionary A^*, then repeated application of the Combined Pivot rule will lead to A^*.

1.4. Reverse Pivoting

Let $\bar{A}$ and A' be two dictionaries with the same dimensions. Suppose application of some pivot rule X to A' yields pivot (s, r) which takes A' to $\bar{A}$. Then we say that (r, s) is a *reverse pivot* for $\bar{A}$ with respect to pivot rule X. In this section we derive necessary and sufficient conditions for a reverse pivot with respect to the Combined Pivot rule. First, the following two lemmas give necessary and sufficient conditions for a reverse pivot with respect to Bland's rule and Dual Bland's rule.

Lemma 1. (r, s) is a reverse pivot with respect to Bland's rule for $\bar{A}$ if and only if the following five conditions hold:

(i) $\bar{a}_{fs} < 0$.

(ii) $\bar{a}_{rs} < 0$.

(iii) $-\dfrac{\bar{a}_{rg}}{\bar{a}_{rs}} = \min \{ -\dfrac{\bar{a}_{ig}}{\bar{a}_{is}} : i \in B - f,\ \bar{a}_{is} < 0 \}$.

(iv) $\bar{a}_{ig} \neq 0$ or $\bar{a}_{is} \leq 0$ for $i \in B - f - r,\ i < s$.

(v) $\bar{a}_{fj} \leq \dfrac{\bar{a}_{fs}\bar{a}_{rj}}{\bar{a}_{rs}}$ for $j \in N - g - s,\ j < r$.

Proof: First we show the necessity of conditions (i)-(v). Suppose (s, r) is selected as a reverse pivot for A'. Then applying Bland's rule to A' and considering the pivot formula we see that:

(a) If $\bar{a}_{fs} \geq 0$ then $a'_{fr} \leq 0$ and s is not selected.

(b) If $\bar{a}_{rs} \geq 0$ then $a'_{rs} \geq 0$ and r is not selected.

(c) If condition (iii) fails, let i be an index achieving the minimum ratio. Then $a'_{ig} < 0$ contradicting the primal feasibility of A'.

(d) If $\bar{a}_{ig} = 0$ and $\bar{a}_{is} > 0$ for $i \in B - f - r$, $i < s$, then the minimum ration in condition (iii) is zero, and $a'_{ir} < 0$. Therefore s is not the smallest row index achieving this ratio.

(e) If condition (v) fails then there is some $j \in N - g - s,\ j < r$ for which $a'_{fj} > 0$. Therefore r is not chosen as the leaving index in A'.

For the sufficiency of the conditions, apply the pivot (r, s) to $\bar{A}$ obtaining dictionary A'. From the pivot formula we see that conditions (i),(ii),(iii) and (v) imply that A' has the sign pattern shown in Figure 1.3.

It is clear that part (1) of Bland's rule selects index r. It remains to see that part (2) selects index s. Let $B' = B - r + s$ denote the basis of A'. Applying the pivot formula, for $i \in B' - s - f$:

$$-\frac{a'_{ig}}{a'_{ir}} = \{ \frac{\bar{a}_{is}\bar{a}_{rg}}{\bar{a}_{rs}} - \bar{a}_{ig} \} / \{ \frac{\bar{a}_{is}}{\bar{a}_{rs}} \} = \bar{a}_{rg} - \frac{\bar{a}_{ig}\bar{a}_{rs}}{\bar{a}_{is}} .$$

and

$$-\frac{a'_{sg}}{a'_{sr}} = \{ \frac{-\bar{a}_{rg}}{\bar{a}_{rs}} \} / \{ \frac{1}{\bar{a}_{rs}} \} = \bar{a}_{rg}.$$

Therefore, since $\bar{a}_{ig} \geq 0$, $\bar{a}_{rs} < 0$ we have

$$\min \{ -\frac{a'_{ig}}{a'_{ir}} : i \in B' - f,\ a'_{is} < 0 \} = \min \{ \bar{a}_{rg},\ \bar{a}_{rg} - \frac{\bar{a}_{ig}\bar{a}_{rs}}{\bar{a}_{is}} : i \in B - f - r,\ \bar{a}_{is} > 0 \} = \bar{a}_{rg}. \quad (1.8)$$

$$
\begin{array}{c|c|c|c|c|}
 & & r & & g \\
\hline
f & \ominus\ \ominus\ \ominus & + & & \\
\hline
 & & & & \oplus \\
 & & & & \oplus \\
 & & & & \oplus \\
\hline
s & & - & & \oplus \\
\hline
 & & & & \oplus \\
 & & & & \oplus \\
 & & & & \oplus \\
\hline
\end{array}
\quad = \quad A'
$$

Figure 1.3 : Sign Pattern of Dictionary A'

($\oplus$ = non-negative entry, $\ominus$ = non-positive entry)

The minimum is clearly unique in (1.8) if $\bar{a}_{ig} > 0$ for $i \in B - f$, in other words if $\bar{A}$ is non-degenerate. In this case, condition (iv) does not apply and part (2) of Bland's rule applied to A' gives index s. Otherwise index s will not be chosen if there is some other minimizer $i < s$, $i \in B - f$ such that $a'_{is} < 0$. But in this case, $\bar{a}_{ig} = 0$ and $\bar{a}_{is} > 0$, which is excluded by condition (iv). □

Lemma 2. (r, s) is a reverse pivot with respect to Dual Bland's rule for a dual feasible dictionary $\bar{A}$ if and only if the following five conditions hold:

(I) $\bar{a}_{rg'} > 0$.

(II) $\bar{a}_{rs} > 0$.

(III) $-\dfrac{\bar{a}_{fs}}{\bar{a}_{rs}} = \min \{ -\dfrac{\bar{a}_{fj}}{\bar{a}_{rj}} : j \in N' - g', \bar{a}_{rj} > 0 \}$.

(IV) $\bar{a}_{fj} \neq 0$ or $\bar{a}_{rj} \geq 0$ for $j \in N' - g' - s, \quad j < r$.

(V) $\bar{a}_{ig'} \geq \dfrac{\bar{a}_{rg'} \bar{a}_{is}}{\bar{a}_{rs}}$ for $i \in B' - f - r, \quad i < s$.

Proof: This can be proved in an analogous way to Lemma 1. Alternatively, if follows from Lemma 1 and the following observation: Applying Dual Bland's rule to a dictionary A is equivalent to applying Bland's rule to the dictionary $-A^T$, where the roles of indices f and g are reversed. □

Note that in view of conditions (ii) and (II), a pivot (r, s) cannot be a reverse pivot with respect to both Bland's rule and Dual Bland's rule. A degenerate optimal dictionary may, however, admit both reverse Bland pivots and reverse Dual Bland pivots. Before stating our main result, we need to restate condition (V) in order to make it more convenient for testing in certain situations.

Lemma 3. Condition (V) of Lemma 2 can be replaced by:

(V')(a) If $\bar{a}_{is} = 0$ then $\bar{a}_{ig'} \geq 0$, for $i \in B' - f, \quad i < s$.

(V')(b) $\dfrac{\bar{a}_{rg'}}{\bar{a}_{rs}} \leq \min \{ +\infty, \dfrac{\bar{a}_{ig'}}{\bar{a}_{is}} : i \in B' - f, i < s, \bar{a}_{is} > 0 \}$.

(V')(c) $\dfrac{\bar{a}_{rg'}}{\bar{a}_{rs}} \geq \max \{ -\infty, \dfrac{\bar{a}_{ig'}}{\bar{a}_{is}} : i \in B' - f, i < s, \bar{a}_{is} < 0 \}$.

Proof: Consider any $i \in B' - f$, $i < s$. If $\bar{a}_{is} = 0$ then clearly (V) and (V')(a) are equivalent. If $\bar{a}_{is} > 0$ then condition (V) becomes

$$\frac{\bar{a}_{ig'}}{\bar{a}_{is}} \geq \frac{\bar{a}_{rg'}}{\bar{a}_{rs}}$$

which is equivalent to (V')(b). Similarly if $\bar{a}_{is} < 0$ condition (V) is equivalent to (V')(c). □

Combining the lemmas we arrive at the following result with is the basis of our implementation.

Theorem 1. (r, s) is a reverse pivot with respect to the Combined Pivot rule if and only if the conditions of either Lemma 1 or Lemma 2 apply. All reverse pivots from a given dictionary can be determined in $O(m(n-m)\ \ min(m, n-m))$ time.

Proof: The first statement follows immediately from the lemmas, so it remains to show how the time complexity is obtained. Suppose we have some given dictionary $\bar{A}$ of dimension $m \times (n-m)$. We first show a complexity bound of $O(m(n-m)^2)$ by processing $\bar{A}$ column by column.

Beginning with Lemma 1, we will fix some column index s and test all possible pivots (r, s), $r \in B - f$. When we begin processing column s, we first compute the set of all indices that obtain the minimum ratio in condition (iii). By using an array of flags, condition (iii) can then be tested in $O(1)$ time per row index. In a similar way, if $\bar{A}$ is degenerate we can compute the smallest index $i \in B - f$ such that $\bar{a}_{ig} = 0$ and $\bar{a}_{is} > 0$, if such an index exists. Now condition (iv) can also be tested in $O(1)$ time per row index. The cost of this preprocessing is $O(m)$ time per column, for a total of $O((n-m)m)$ time for the entire dictionary.

With the above preprocessing, the only condition requiring more than $O(1)$ time is condition (v). This condition may require $O(n-m)$ time per candidate. Note, however, that it is not performed if any of the previous tests fails. Since there are $m(n-m)$ candidates, the complexity of testing for all reverse pivots from a given dictionary is $O(m(n-m)^2)$.

Now consider testing the conditions of Lemma 2. Conditions (I), (II) can be tested in $O(1)$ time, and conditions (III), (IV) can be tested in $O(n-m)$ per candidate (r, s). It remains to show how condition (V) can be efficiently tested. For this we use the alternate condition given in Lemma 3. For fixed column s, we first test condition (V')(a). If this fails there can be no reverse pivot using this column. Otherwise, the right hand sides of conditions (V')(b) and (V')(c) are computed in $O(m)$ time and stored. Then for each row index r, the condition (V') can be tested in $O(1)$ time.

The above argument shows that if $\bar{A}$ is processed column by column, all reverse pivots can be found in $O(m(n-m)^2)$ time. However, in case $m < n-m$ it pays to process the dictionary row by row. In this case the method used for testing Lemma 1 above is essentially used for testing Lemma 2, and vice versa. Condition (v) of Lemma 1 needs to be rewritten in a similar way as Condition (V) of Lemma 2. This gives a complexity of $O(m^2(n-m))$ for finding all reverse pivots from $\bar{A}$. The details are omitted. □

The above complexity result is an improvement on the complexity of $O(nm(n-m))$ reported in AF. For example, for the vertex enumeration problem of section 1.2, in fixed dimension d the above method has complexity $O(n_0)$ per basis, compared to $O({n_0}^2)$ per basis in AF. Also, as in AF if the optimum dictionary is unique, the complexity is bounded by $O(n(n-m))$ for non-degenerate pivots. For in this case Lemma 2 never applies, and we know that the minimum in condition (iii) of Lemma 1 is unique. Therefore, condition (v) of Lemma 1 is only tested once per candidate column. Conditions (iii) and (v) therefore take a total of $O(n)$ time per column.

It seems likely that a more sophisticated analysis may give an amortized worst case time bound of $O(m(n-m))$ even in the degenerate case. Limited computational experience shows that condition (v) is usually not required to reject an invalid candidate for a reverse pivot. For example, on a 9-dimensional problem with 14 hyperplanes and a unique optimal dictionary, 4086 reverse Bland pivots were tested. Of these, 2170 were rejected by condition (i), 1005 by condition (ii), 498 by condition (iii) and 1 by condition (iv). Condition (v) was therefore executed only 316 times, and a total of 943 coefficients of row zero were generated. The polyhedron has 89 vertices and 97 feasible bases, and its graph and reverse search tree are shown in Figure 3.5.

2. Implementation

The reverse search vertex enumeration algorithm was implemented in ANSI C and is named *ve*04. *c*. This section refers to Version 0.4, released March 31, 1993. This program and some sample input files can be obtained by anonymous *ftp* from mutt.cs.mcgill.ca (132.206.3.13). Please use login name "ftp" and change to directory pub/C. To run, simply compile the file *ve*04. *c* and run the a.out file. A runtime option of 0 (eg. a.out 0) will cause the program to print all *bases*, rather than all *vertices*. For the distinction, see the paragraph on degeneracy later in this section. Most users will simply want all vertices. Please report any bugs to the author at avis@cs.mcgill.ca.

The main changes with release 0.4 are the ability to restart the program from any basis and new *divide* and *gcd* procedures coded by Jerry Quinn. The *divide* routine in previous versions contains a bug and occasionally gives incorrect results. Version 0.4 is roughly twice as fast as earlier versions. Input files for previous versions require a small modification: append a line with a single "n", indicating that the program is not restarting in the middle of a calculation (see Figure 3.1). To restart from a known basis, answer "y" to the question "Restarting from known cobasis? (y/n)" and then enter the vertex number, basis number, depth and cobasis indices for the restart (see Figure 3.2).

Unlimited precision rational arithmetic was employed to avoid numerical problems. In this section we describe the data structure and structure of procedure *main* which drives the reverse search. The major functions called from *main* are described briefly.

The dictionary $\bar{A}$ contains most of the data in *ve*04. *c*. Predefined constants M and N bound the maximum number of rows and columns of $\bar{A}$. These may be modified by the user to fit the requirements of the problem to be solved. They are currently set to 40 each respectively. Each entry in $\bar{A}$ is a rational number, which is stored as a pair of extended precision integers, one for the numerator and one for the denominator. These integers are in turn each stored as an array, indexed (by C convention) $0,1,...,DIGITS-1$, where $DIGITS$ is a predefined constant, currently set at 201. Let a denote such an array. In index zero we store the length, $length(a)$ and the sign, $sign(a)$ of the integer. The integer itself is stored in positions $1,2,...,length(a)$. Each position of a contains a "digit" of the integer in base $BASE$, with low order digits first. On 32-bit machines it is convenient to set $BASE = 10000$ so each position of a contains in fact 4 decimal digits. Therefore, with DIGITS=201 up to about 800 digits can be handled. As an example, the integer -271828182845904 is stored as $a[0] = -4$, $a[1] = 5904$, $a[2] = 8284$, $a[3] = 8281$, $a[4] = 271$. This data structure is described in Gonnet and Baeza-Yates[6] and is defined as type *mp* in *ve*04. *c*. From this handbook we use basic routines to add, subtract and multiply to extended precision integers. The division routine was coded by Jerry Quinn and is based on Knuth[7]. On top of these basic functions we use a standard Euclidean algorithm for computing the greatest common divisor. With these functions it is straight forward to implement rational arithmetic.

The parameter DIGITS greatly effects the storage required, since the main array uses N*M*DIGITS integers. DIGITS can be reset by the user to any desired value. To get an upper bound on the maximum precision needed in the calculation we can use the well known Hadamard inequality (see for example[7]). If $b = (b_{ij})$ is an m by m matrix, then

$$|det(b_{ij})| \le \prod_{1 \le i \le m} \left(\sum_{1 \le j \le m} a_{ij}^2 \right)^{1/2} .$$

For example, if a is $\bar{A}$ is integral with no entry larger than 10^y then the largest m by m determinant is at most $m^{m/2}\ 10^{my}$, so $my + 0.5\ m \log_{10} m$ digits of precision are enough. To allow for intermediate results, four times this many digits should be allocated.

From the above description, we see that the dictionary $\bar{A}$ is actually stored as two *three*-dimensional arrays: a numerator array *NA* and a denominator array *DA*. The entries $NA[i][j]$ and $DA[i][j]$ are each one-dimensional arrays containing respectively the numerator and denominator of $\bar{a}_{ij}$. Along with the dictionary $\bar{A}$, we store two additional one-dimensional (integer) arrays B and C, storing, respectively, the basic and co-basic indices.

In Section 1.2 we showed how to construct the initial dictionary $\bar{A}$ for the vertex enumeration problem for a given polyhedron P. Recall, the variables are indexed $0,1,...,n-1$ and the dictionary has m rows and $n-m$ columns. Section 1.2 also gives the initial values for B and C. A pivot interchanges an index of B and C. Since we would like to keep these arrays sorted we have two options: reorder the rows and columns of the dictionary after each pivot, or keep track of the location (row or column index) of each variable. The second option was chosen as it is computationally faster, employing an additional array *LOC* to store locations. In this way, $B[i]$ contains the ith smallest index in the basis, and $LOC[B[i]]$ contains the row index of *NA* and *DA* where the corresponding row of dictionary $\bar{A}$ is stored. Similarly, $C[j]$ contains the jth smallest index in the co-basis, and $LOC[C[j]]$ contains the column index of *NA* and *DA* where the corresponding column of dictionary $\bar{A}$ is stored.

One additional array is used to speed up the reverse pivot calculations. As described in the proof of Theorem 1, a boolean array *minratio* of flags is computed each time we start checking for a reverse pivot from some new column. The flag *minratio*$[i]$ is *TRUE* if the variable with index $B[i]$ obtains the minimum ratio for the column. Finally a number of flags are kept to record degeneracy, feasibility etc. The entire global data base is given in Figure 2.1.

```
#define DIGITS 201                  /*number of digits of extended precision      */
#define TRUE 1
#define FALSE 0
#define BASE 10000
#define M 40                        /* max number of rows in matrix               */
#define N 40                        /* max number of cols in matrix               */
typedef int mp[DIGITS];             /* type mp holds one multi-precision integer  */

mp NA[M+1][N+2],DA[M+1][N+2];       /* Hold numerator and denominator of array A  */
int m=0,n=0;                        /* actual dimensions of matrix A are m by n-m */
int g=0;                            /* g=n-m-1 is index of RHS                    */
int last;                           /* last=g if opt is non-degen, else=n-m       */
int B[M+1], C[N+2];                 /* hold basis and co-basis indices in order   */
int LOC[N+M+3];                     /* location of given index 0..m-1 are rows,   */
                                    /* 0..last are cols of A                      */
int degenerate=FALSE;               /* TRUE if current dictionary is degenerate   */
int ddegenerate=FALSE;              /* TRUE if current dictionary is dual degen.  */
int degenopt=FALSE;                 /* TRUE if optimal dictionary is degenerate   */
int minratio[M+1];                  /* TRUE if B[i] obtains min ratio else FALSE  */
int dminratio[N+2];                 /* TRUE if C[j] obtains min ratio else FALSE  */
int ratios=FALSE;                   /* TRUE if ratios calculated for the current col */
int dratios=FALSE;                  /* TRUE if ratios calculated for the current row */
int dual=TRUE;                      /* TRUE if dual feasible, ie. row 0 negative  */
```

Figure 2.1 Global Data Base

3. Computational Experience

In this section we give a sample run and some preliminary computational results. Consider the following three dimensional polyhedron P:

$$x_1 - 2x_2 + 3x_3 \le 1$$

$$-4x_1 + 5x_2 - 6x_3 \le 2$$
$$7x_1 - 8x_2 + 9x_3 \le 3$$
$$x_1 \ge 0, \quad x_2 \ge 0, \quad x_3 \ge 0$$

Figure 3.1 gives a sample run for for P. The output gives the values of the variables at each vertex. The original variables of the problem appear as x[1],x[2],x[3]. In addition a slack variable is added for each inequality, these appear, respectively, as x[4],x[5],x[6]. Normally the user is only interested in the original variables. For each vertex, the program prints the values of the basic variables. The cobasic variables are all zero, and there indices are listed. For example, at Vertex #1 all of the original variables have value zero. Vertex #3 corresponds to the vertex with $x_1 = 0$, $x_2 = 4$ and $x_3 = 3$. There is some degeneracy in this problem as we observe that there are 6 vertices and at least 8 bases: there may be some additional bases after Basis #8 but no new vertices.

```
INPUT:

>Vertex Enumeration for Polyhedron { Ax <=b ,x>=0 }
>A is an m by n rational matrix, b is a non-negative m vector

>Enter number of rows(max=40) and columns(max=40) of A
3 3

>Enter Numerator Matrix (with optional sign)

Row 1 =  1 -2   3
Row 2 = -4  5  -6
Row 3 =  7 -8   9

>Enter Numerator of Positive b Vector
b[1] = 1
b[2] = 2
b[3] = 3

>Denominators all ones? (y/n)
y

>Restarting from known cobasis? (y/n):
n

OUTPUT:

 Basis    0 4 5 6  Location 0 1 2 3
 Co-Basis 1 2 3  Location 0 1 2   dual feasible
A[0][2]=-2/1  [4]=-1/6   [6]=1/6   [7]=-1/3
A[1][2]=1/2   [4]=3/4    [6]=-1/4  [7]=0/1
A[3][2]=1/2   [4]=-7/12  [6]=1/12  [7]=1/3
A[5][2]=0/1   [4]=-1/2   [6]=-1/2  [7]=4/1

 Vertex #1   Basis # 1  Depth=0 Entering=0 Co-Basis 1 2 3
x[4]=1/1      x[5]=2/1      x[6]=3/1

 Vertex #2   Basis # 2  Depth=1 Entering=2 Co-Basis 1 3 5
x[2]=2/5      x[4]=9/5      x[6]=31/5

 Vertex #3   Basis # 3  Depth=2 Entering=3 Co-Basis 1 4 5
x[2]=4/1      x[3]=3/1      x[6]=8/1

 Vertex #4   Basis # 5  Depth=2 Entering=1 Co-Basis 2 4 6
x[1]=0/1      x[3]=1/3      x[5]=4/1

 Vertex #5   Basis # 7  Depth=2 Entering=1 Co-Basis 2 3 6
x[1]=3/7      x[4]=4/7      x[5]=26/7

 Vertex #6   Basis # 8  Depth=3 Entering=2 Co-Basis 3 5 6
x[1]=31/3    x[2]=26/3    x[4]=8/1
```

Figure 3.1   Sample Run

In Figure 3.2 we show how to restart the program from a known vertex. This is particularly useful if there is a machine shutdown (or power failure!) during a long computation. In the figure we restart from Vertex #4 Basis #5 which is at Depth=2 in the tree. To restart, use the same input file as for a regular run except answer "y" to the question "Restarting from known cobasis? (y/n)" and then enter the vertex number, basis number, depth and cobasis indices for the restart.

```
INPUT:

>Vertex Enumeration for Polyhedron { Ax <=b ,x>=0 }
>A is an m by n rational matrix, b is a non-negative m vector
>Enter number of rows(max=40) and columns(max=40) of A:

********Same as Figure 3.1 ***************


Restarting from known cobasis? (y/n): y

Enter Vertex # or zero: 4

Enter Basis # or zero: 5

Enter Depth or zero: 2

Enter Co-Basis for restart: 3 indices between 1 and 6: 2 4 6

Restarting from Vertex # 4  Basis # 5  Depth=2  Co-Basis 2 4 6
 Basis    0 1 3 5  Location 0 1 3 2   degenerate
 Co-Basis 2 4 6  Location 1 0 2   dual infeasible
A[0][2]=-2/1  [4]=-1/6   [6]=1/6   [7]=-1/3
A[1][2]=1/2   [4]=3/4    [6]=-1/4  [7]=0/1
A[3][2]=1/2   [4]=-7/12  [6]=1/12  [7]=1/3
A[5][2]=0/1   [4]=-1/2   [6]=-1/2  [7]=4/1

 Vertex #4   Basis # 5  Depth=2 Entering=0 Co-Basis 2 4 6
x[1]=0/1       x[3]=1/3       x[5]=4/1

 Vertex #5   Basis # 7  Depth=2 Entering=1 Co-Basis 2 3 6
x[1]=3/7       x[4]=4/7       x[5]=26/7

 Vertex #6   Basis # 8  Depth=3 Entering=2 Co-Basis 3 5 6
x[1]=31/3      x[2]=26/3      x[4]=8/1
```

Figure 3.2 Restart of Sample Run from Vertex 4

Some preliminary computational experience is given in Figure 3.3. These examples contain both simple polyhedra and highly degenerate polyhedra. Problem 1 had 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 on these four points. Problem 4 and 5 were constructed arbitrarily with integer data in the range -100..100. The reverse search tree generated for problem 4 and the graph of its polyhedron is shown in Figure 3.5. 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. All times shown are for a SPARC station 1.

Problem	No. of Hyperplanes $n_0 + d$	Dimension d	No. of Vertices	No. of Bases	Time seconds
1	34	4	31	31	29.3
2	16	5	18	1247	34.2
3	19	6	8	10845	398.3
4	12	7	54	54	1.2
5	14	9	89	97	3.3
6	23	10	332	3656	282.2
7	20	10	1188	1188	3596.0

Figure 3.3 Computational Results

The test run for Problem 5 was run with the C profiler *gprof* to see where the execution time is spent. With the profiler, the running time was 9.25 seconds distributed as shown in Figure 3.4.

Procedure	Time (seconds)	No. of times called
pivot	7.17	192
reverse	1.10	4365
selectpivot	0.12	96
printsol	0.02	97
update	0.02	192
increment	0.00	4086

Figure 3.4 Execution Profile for Problem 5

From this profile we see that about 84% of the execution time is spent in pivoting and only about 13% in testing for reverse pivots, using the ideas of Theorem 1. Earlier implementations without Theorem 1 used substantially more time in testing reverse pivots than in pivoting. The number of pivots made is always 2*(number of bases - 1) using the reverse search method. This suggests further significant improvements in running time can only be made by improving the low level multi-precision arithmetic procedures used in pivoting.

Finally we briefly discuss the problem of degeneracy. A degenerate problem is a problem where for some vertex or vertices many more than d inequalities are satisfied as equations. If $k > d$ such equations exist for a given vertex, as many as $\binom{k}{d}$ bases may represent it. The reverse search method will evaluate all of these bases, although only one (the lexicographically minimum) will be output. For highly degenerate problems it can be of great advantage to perturb the right hand side slightly by adding or subtracting small randomly chosen numbers. If the resulting polyhedron is non-degenerate the total number of vertices is governed by the Upper Bound Theorem of P.McMullen (see[3]). This states that the maximum number of vertices of any polyhedron defined by n_0 inequalities in d non-negative variables is

$$\binom{n_0 + \lfloor \frac{d}{2} \rfloor}{n_0} + \binom{n_0 + \lfloor \frac{d-1}{2} \rfloor}{n_0}.$$

This is then an upper bound on the output size for a non-degenerate problem. In the extreme case a completely degenerate problem, in which each inequality is bounded by a hyperplane passing through the origin, could have the roughly quadratically larger $\binom{n_0 + d}{d}$ number of bases. Therefore, perturbation is greatly recommended.

4. Future Work

The present implementation can be improved in many ways. At present it does not achieve the theoretical complexity of $O(m(n-m)\ min(m,\ n-m))$ because it always processes reverse pivots in column order, and the speed up for condition (V') is not implemented. Hence the current implementation has complexity $O(m(n-m)^2)$ if the optimum dictionary is unique, and otherwise $O(m(n-m)n)$. This is unlikely to matter in practice unless the input problem has few rows but many columns. It may not matter at all if the amortized cost per candidate reverse pivot is in fact constant, as suggested by experiment.

With the restart feature, the current code is suitable for distributed processing. It is easy to modify the code to compute a reverse search tree down to some preset depth k. Each node at level k can then be treated as an independent problem and assigned to a separate processor. Using the depth counter, these processes can be programmed to stop after they have completely enumerated the sub-tree for which the given startup node is the root. It is planned to formalize this with some standard protocol for distributed computing. Various options for the program should be incorporated as command line arguments. Finally, it is planned to incorporate the current code into the *Mathmatica* package[5].

5. Acknowledgements

The author gratefully acknowledges frequent discussions with Komei Fukuda during the course of this work. He also kindly provided Figure 3.5. Most of the programming work was carried out in the laboratory of Professor Masakazu Kojima at Tokyo Institute of Technology, and was supported by the N.S.E.R.C./J.S.P.S. bilateral exchange program. Akihisa Tamura provided some highly degenerate test problems that uncovered several bugs in the code. The author is extremely grateful for the excellent facilities that were made available, and for the assistance of many members of the laboratory. Version 0.4 contains new divide and gcd routines coded by Jerry Quinn at McGill. He also made a number of other improvements to speed up the code.

This paper is a revised version of Technical Report SOCS-92.12, McGill University, School o f Computer Science, October 1992.

References

1. 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.
2. R. G. Bland, "A Combinatorial Abstraction of Linear Programming," *J. Combin. Theory B*, vol. 23, pp. 33-57, 1977.
3. V. Chvátal, *Linear Programming,* W.H. Freeman, 1983.
4. H. Edelsbrunner, *Algorithms in Combinatorial Geometry,* Springer-Verlag, 1987.
5. K. Fukuda and I. Mizukoshi, *Mathematica Package: Vertex Enumeration for Convex Polyhedra and Arrangements,* Version 0.3 Beta, Graduate School of Systems Management, University of Tsukuba, Tokyo.

6. G.H. Gonnet and R. Baeza-Yates, *Handbook of Algorithms and Data Structures-2nd Edition,* Addison-Wesley, 1991.

7. D. E. Knuth, *The Art of Computer Programming, Vol 2: Seminumerical Algorithms,* Addison-Wesley, Reading, MA, 1981.