# Computational Experience with the Reverse Search Vertex Enumeration Algorithm

*David Avis*

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

January 26, 1999

Dedicated to Professor Masao Iri on the occasion of his 65th birthday

*ABSTRACT*

This paper describes computational experience obtained in the development of the *lrs* code, which uses the reverse search technique to solve the vertex enumeration/convex hull problem for $d$-dimensional convex polyhedra. We give empirical results showing improvements obtained by the use of lexicographic perturbation, lifting, and integer pivoting. We also give some indication of the cost of using extended precision arithmetic and illustrate the use of the estimation function of *lrs*. The empirical results are obtained by running various versions of the program on a set of well-known non-trivial polyhedra: cut, configuration, cyclic, Kuhn_Quandt, and metric polytopes.

Keywords: vertex enumeration, convex hulls, reverse search, computational experience

## 1. Introduction

A classic result is that a convex polyhedron $P$ can be represented in two ways. An *H-representation* is given by an $m \times d$ matrix $A = (a_{i,j})$ and $m$-vector $b = (b_i)$:

$$P = \{ y \in R^d \mid b + Ay \geq 0 \} \qquad (1.1)$$

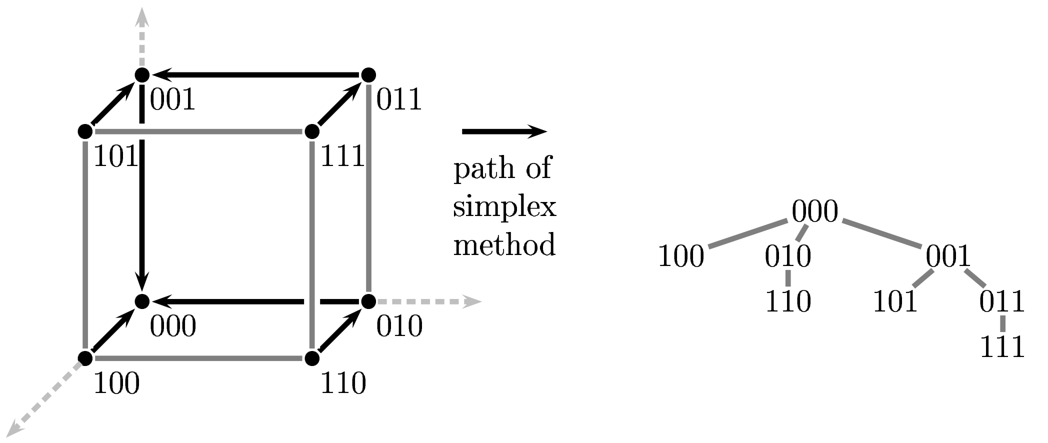
If $A$ is minimal, that is no row can be deleted without changing $P$, then $P$ has $m$ *facets*, each defined by one of the inequalities in (1.1). A *vertex* $y \in R^d$ is a point of $P$ that satisfies an affinely independent set of $d$ inequalities as equations. We assume throughout that $P$ has at least one vertex, which implies that $m \geq d$. 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* $z_1, \cdots, z_u$:

$$P = \{ y \in R^d \mid y = \sum_{i=1}^{s} \lambda_i y_i + \sum_{j=1}^{u} \mu_j z_j, \lambda_i \geq 0, \mu_j \geq 0, \sum_{i=1}^{s} \lambda_i = 1 \}. \qquad (1.2)$$

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.

This paper describes computational experience obtained in the development of the *lrs* code [1], to solve the vertex/facet enumeration problem. A complete technical description of the methods described here is contained in the companion paper [8]. The code is based on the reverse search algorithm proposed by Fukuda and the author [4]. 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. 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 each vertex lies on exactly $d$ hyperplanes, and so has a unique basis. In this case the polyhedron is called *simple* or *non-degenerate*. 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 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: Reverse Search Tree for Vertex Enumeration of the Cube**

In this paper we discuss various implementations of this algorithm: *ve*01, *ve*06, *qrs*, and *lrs*. The first implementation, *ve*01 was released by the author in 1992, and revised in 1994[6]. It solved the vertex enumeration problem for *polytopes* (bounded polyhedra), and did not report extreme rays. A technical description of the latest implementation, *lrs*, is given in[8]. All of the implementations use extended precision exact arithmetic. A summary of differences between the various codes is given in Table 1.

In [6] some preliminary computational experience was given on a set of seven test problems. Table 2 shows the improvement in running time obtained between the original program *ve*01 and version 2.3 of *lrs* on these test problems (for information on the machines used for the experiments in this paper, see the appendix). All of the problems *in*1,...,*in*7 are vertex enumeration problems, with #*F* input inequalities in $d$ dimensions. In the table, #V and #R refer respectively to the number of vertices, and rays computed. Evidently *ve*01 computes many more bases

| Code | polytope/polyhedron | perturbation | lifting | arithmetic | |
|---|---|---|---|---|---|
| | | | | rational/integer | extended precison |
| *ve01* | polytope | - | - | R | yes |
| *ve06* | polytope | random number | - | R | yes |
| *qrs* | polytope | random number | - | I | yes |
| *lrs* | polyhedron | lex-positive | yes | I | yes |

**Table 1: Features of the Codes Described**

than *lrs* in some cases, and this will be discussed in Section 3.1.

| Problem | facets | dim. | vertices | rays | *ve01* | | *lrs* | | *ve01/lrs* |
|---|---|---|---|---|---|---|---|---|---|
| | #F | d | #V | #R | bases | secs | bases | secs | time |
| in1 | 34 | 4 | 31 | | 31 | 2.81 | 31 | .06 | 46 |
| in2 | 16 | 5 | 18 | | 1247 | 7.48 | 76 | .05 | 148 |
| in3 | 19 | 6 | 8 | | 10845 | 86.43 | 188 | .12 | 720 |
| in4 | 12 | 7 | 54 | | 54 | .43 | 54 | .05 | 9 |
| in5 | 14 | 9 | 89 | 33 | 97 | 1.14 | 94 | .09 | 13 |
| in6 | 23 | 10 | 332 | 302 | 3656 | 83.65 | 824 | 1.69 | 50 |
| in7 | 20 | 10 | 1188 | | 1188 | 208.98 | 1188 | 2.78 | 75 |

**Table 2: Computational Results on Original Problem Set (*ve01 vs lrs*)**

The purpose of this paper is to quantify empirically various improvements made, in order to draw some conclusions that may be useful to other geometric computational problems. In particular, we study the use of perturbation, lifting, and integer pivoting. We give some indication of the cost of using extended precision arithmetic by comparing running times with a fixed 64-bit integer version of the code. Finally we illustrate the estimation functions of *lrs*. In the next section we give a new set of test problems that are considerably more difficult than the set discussed above.

## 2. Test Problems

We study the behaviour of various versions of the reverse search code on a set of five classes of polyhedra. Although *lrs* handles unbounded polyhedra and does facet enumeration, earlier versions of the code did not report extreme rays, and performed vertex enumeration only. To facilitate comparisons between versions, we choose examples that are polytopes. Since a facet enumeration problem can be solved directly as a vertex enumeration problem for polytopes containing the origin as an interior point, we restrict our facet enumeration test problems to these polytopes. The problems were chosen to include both simple and highly degenerate polytopes, and to require a wide range of digits of precision in the calculations. Each example comes from a class of polytopes with a large literature. We describe them briefly here. Each problem is given a name *name m_n*, where $m$ and $n$ denote the size of the input array for the problem. For vertex enumeration, $m$ is the number of input inequalities, and for facet enumeration it is the number of vertices. $n$ is the dimension of the polytope plus one. Data for the seven problems chosen is given in Table 3, and the input files are available on-line as described in the appendix. The first three problems and the last one are facet enumeration problems, and the others are vertex enumeration problems. For each problem the output to be computed is highlighted in bold face. The volume is given for the facet enumeration problems. Although the volume is computed exactly, a floating point approximation is given here due to the large size of the integers.

### 2.1. Cut Polytope

For $n \geq 4$, let $x = (x_{i,j})\ 1 \leq i < j \leq n$ be a vector of length $n(n-1)/2$. For convenience, we identify $x_{i,j}$ and $x_{j,i}$. The cut polytope is given by the convex hull of the following $2^{n-1}$ vertices. For every subset $S$ of $\{1, 2, \ldots, n-1\}$ define

| Problem | input rep. | dim. d | facets #F | vertices #V | volume | *lrs* bases | *lrs* secs |
|---|---|---|---|---|---|---|---|
| cut16_11 | V | 10 | **56** | 16 | 2.3116... | 124 | 0.13 |
| cut32_16 | V | 15 | **368** | 32 | 1.3457... | 46751 | 169.67 |
| cyclic25_13 | V | 12 | **35700** | 25 | 4.84e+75 | 18564 | 909.23 |
| kq20_11 | H | 10 | 20 | **1188** | - | 1188 | 3.84 |
| metric40_11 | H | 10 | 40 | **32** | - | 9184 | 17.78 |
| metric80_16 | H | 15 | 80 | **544** | - | 6450702 | 37532.68 |
| mit31_20 | V | 19 | **18553** | 31 | 1.712e+3 | 36354 | 326.37 |

**Table 3: New Test Problems**

$$x_{i,j} = \begin{cases} 1 & \text{if precisely one of } i \text{ and } j \in S \\ -1 & \text{otherwise} \end{cases}$$

In the standard definition of a cut polytope, the -1 is replaced by zero. We use the above formulation so that the polytope contains the origin. These polytopes are described in detail in the book of Deza and Laurent[14]. The description above is a $V$-description, so we consider facet enumeration for the cut polytopes with $n = 5, 6$ named *cut*16_11 and *cut*32_16 respectively.

## 2.2. Metric Polytope

Let $n$ and $x$ be defined as above. The metric polytope is given by the system of inequalities

$$x_{i,j} - x_{j,k} - x_{i,k} \le 0$$

$$x_{i,j} + x_{j,k} + x_{i,k} \le 2$$

for every set of distinct indices $i, j, k$ in the range $1, \ldots, n$. Note that for each triple there are three inequalities of the first type and one of the second. The metric polytope is closely related to the metric cone, which can be used to determine infeasibility of fractional multicommodity flows, see Iri[19]. Early work on the extreme rays of the metric cone is contained in Avis[3] and Lomonosov[22]. Up to translation and scaling, the vertices of the cut polytope are vertices of the metric polytope. They describe the same polytope when $n = 4$, otherwise the cut polytope is a strict sub-polytope. The description above is an $H$-description, so we consider facet enumeration for the metric polytopes with $n = 5, 6$ named *metric*40_11 and *metric*80_16 respectively.

## 2.3. Cyclic Polytope

Cyclic polytopes achieve the bound of requiring the most facets of any polytope with the same dimension and number of vertices, see for example Ziegler[23]. They can be defined in any dimension $d \ge 2$ by choosing $n > d$ vertices of the form $(t, t^2, \cdots, t^d)$, for $n$ distinct values of $t$. Cyclic polytopes are notoriously unstable numerically and require very long integers when manipulated by programs using exact arithmetic. We consider the facet enumeration problem for the example *cyclic*25_13 with 25 vertices in 12 dimensions defined by the integers $-12 \le t \le 12$. The polytope was translated and scaled to contain the origin and maintain integer coordinates.

## 2.4. Configuration Polytope

Configuration polytopes arise in the modelling of alloys. They were brought to our attention by Ceder and Garbulsky[12] and provided the first large problem solved by an early implementation of *lrs*: a polytope with 729 inequalities in 8 dimensions and 4,862 facets which in 1992 took over a month to solve. Even after perturbation the polytope contained 477,421 bases. Here we consider a smaller example, *mit*31_20 which is a facet enumeration problem with a $V$-representation with 31 vertices in 19 dimensions. The input matrix is quite sparse, and contains integers between -12 and 12.

## 2.5. Kuhn-Quandt Polytopes

Kuhn-Quandt polytopes are defined by random matrices, and historically were one of the first test problems for evaluating variants of the simplex method[21]. The problems have the form $Ax \leq b$, $x \geq 0$, with matrix entries randomly chosen in the range 0..1000 and $b$ vector entries all 10000. In our example, *kq*20_11 (identical to *in7* in Table 2), we choose a 10 by 10 matrix with these properties, and add 10 non-negativity constraints.

## 3. Speedups

In this section we discuss methods used to speed up the original implementation of reverse search. Unless otherwise stated, we will refer to the vertex enumeration problem, where the input is an $H$-description and the required output is a $V$-description.

## 3.1. Perturbation

Pivoting methods for vertex enumeration enumerate bases for the vertices in the $V$-description, as described in the introduction. Among the test problems, *cyclic*25_13 and *kq*20_11 are simple polyhedra, but the other problems are highly degenerate, making a computation of all bases extremely expensive. Using various implementations of reverse search, we can study the effect of perturbation to reduce the number of bases. In this section, we treat all problems as $H$-representations to facilitate the use of earlier codes. For the four facet enumeration problems in Table 3, this corresponds to doing a vertex enumeration of the dual polytope. The number of bases in each unperturbed polytope is given in column 4 of Table 4.

| Problem | input | vertices | bases | | |
|---|---|---|---|---|---|
| | rep. | #V | unperturbed | perturbed | lex-positive |
| cut16_11* | H | 56 | 1936 | 496 | 496 |
| cut32_16* | H | 368 | 44450496 | 264612 | 186138 |
| cyclic25_13* | H | 35700 | 35700 | 35700 | 35700 |
| kq20_11 | H | 1188 | 1188 | 1188 | 1188 |
| metric40_11 | H | 32 | 115972624 | 9254 | 9184 |
| metric80_16 | H | 544 | ? | 7762890 | 6450702 |
| mit31_20* | H | 18553 | 16184439 | 168300 | 169272 |

**Table 4: Perturbation (** = dual polytope*)**

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 *ve*06 and *qrs* by adding a small random number to the $b$ vector. For the degenerate test problems we added a rational $t/10000$ where $t$ was a randomly chosen integer in the range 1..100, and used the seed 1234 for the random number generator. Column 5 of Table 4 shows the result of the perturbation, giving the number of bases of the perturbed polytope that are computed if either *ve*06 or *qrs* is used.

Numerical perturbations creates several difficulties: (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. For this reason it was abandoned. *lrs* resolves degeneracy by use of the well-known lexicographic pivot selection rule for the simplex method (see for example, Ignizio and Cavalier [18]. ) 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. It is known that the set of lex-positive bases of a polytope $P$ in fact is combinatorially equivalent to the set of bases of a certain numeric perturbation of $P$. The last column of Table 4 shows the number of lex-positive bases for each problem. We remark that this number in general depends on the order of the input data. For the original set of test problems in

Table 2, column 6 gives the number of bases of the input polyhedron, which are computed by *ve*01. Column 8 gives the number of lex-positive bases computed by *lrs*. Since the running time of reverse search methods is proportional to the number of bases computed, the speedup for the highly degenerate problems is very large.

Lexicographic perturbation has several other advantages. 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. This considerably simplified the implementation, as the earlier versions required a preliminary phase of reversing the dual simplex method in order to generate all optimal dictionaries. With suitable labelling, the lex-min basis for each vertex is lex-positive and this property can be tested quickly. By reporting only lex-positive bases we avoid output duplication for degenerate inputs. Finally, for the facet enumeration problem, lexicographic perturbation induces a triangulation of $P$, which can be used to enable the volume to be computed readily. A detailed description of the underlying theory is given in[8].

## 3.2. Lifting

The programs *ve*01, *ve*06 and *qrs* perform only the transformation of an $H$-representation to a $V$-representation. In order to allow the reverse transformation, the facet enumeration problem, a standard lifting technique ( see for example[23], Chapter 1 ) was implemented in *lrs*. The input $V$-representation is lifted (or homogenized) 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

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

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 $P$. It is also possible to homogenize an $H$-representation. Each inequality

$$b_i + a_{i,1} y_1 + \cdots + a_{i,d} y_d \geq 0$$

is replaced by an inequality

$$b_i y_0 + a_{i,1} y_1 + \cdots + a_{i,d} y_d \geq 0$$

and the additional inequality $y_0 \geq 0$ is added. Again we obtain a pointed cone in one higher dimension.

The lifted polyhedron has one completely degenerate vertex, the origin. Since pivoting methods perform badly on degenerate polyhedra, it was expected that lifting would result in slower running times. It came as a surprise that in fact the reverse seems to be the case. As mentioned in the introduction to Section 2, the facet enumeration problem for polytopes containing the origin can be solved as a vertex enumeration problem. In order to test *lrs*, we compared the output obtained in this way with the results obtained by lifting, see Table 5. We ran each problem with the input as an $H$-description, getting the unlifted results, and as a $V$-description getting the lifted results. In all cases the lifted polytope contains fewer bases, and speedups of the order of 2-4 times were obtained.

These results can be explained by the fact a given dictionary can represent only one vertex, but may represent many rays (up to the number of columns $d$). Rays are easily detected by checking the signs of the current dictionary, and can be output immediately. For each vertex, however, a pivot is required to compute its dictionary. In the case where this dictionary is a leaf of the reverse search tree, this pivot is "wasted", as is the time required to determine this fact.

| Problem | bases computed | | seconds | | unlifted/lifted | |
|---|---|---|---|---|---|---|
| | unlifted | lifted | unlifted | lifted | bases | time |
| cut16_11 | 496 | 124 | 0.53 | 0.13 | 4.00 | 4.08 |
| cut32_16 | 186138 | 46751 | 639.61 | 169.67 | 3.98 | 3.76 |
| cyclic25_13 | 35700 | 18564 | 1799.18 | 909.23 | 1.92 | 1.98 |
| kq20_11 | 1188 | 588 | 3.84 | 2.03 | 2.02 | 1.89 |
| metric40_11 | 9184 | 2304 | 67.02 | 4.96 | 3.99 | 3.59 |
| metric80_16 | 6450702 | 1588778 | 37532.68 | 12114.82 | 4.06 | 3.08 |
| mit31_20 | 169272 | 36354 | 1477.82 | 326.37 | 4.66 | 4.53 |

**Table 5: Lifting (*lrs*)**

## 3.3. Rational vs Integer Arithmetic

All implementations described in this paper use exact integer arithmetic. The extended integers are stored in arrays of long integers. For 32-bit (respectively, 64-bit) machines, each array element stores 4 (respectively, 9) decimal digits. The basic arithmetic and normalization functions were taken from Gonnet and Baeza-Yates[16], except for the long division routine which was implemented by Quinn based on Knuth[20], p. 320.

The programs *ve*01 and *ve*06 were implemented in rational arithmetic. The dictionary was saved in two extended integer matrices, one for the numerators and one for the denominators. After each arithmetic operation involving two rational numbers, the resulting rational is reduced by dividing by the greatest common divisor (gcd) obtained by Euclid’s algorithm. For integer arithmetic, division is the most expensive operation. For rational arithmetic, division and multiplication are equivalent requiring two multiplications and a divide. They are faster than addition/subtraction which requires 3 multiplications, one addition/subtraction and one divide.

The most time consuming operation in all implementations is pivoting the dictionary. To pivot a rational dictionary on row $r$ and column $s$ the following operations are performed:

$$\bar{a}_{i,j} = a_{i,j} - \frac{a_{i,s}a_{r,j}}{a_{r,s}} \tag{3.1}$$

$$\bar{a}_{r,s} = \frac{1}{a_{r,s}}\ , \quad \bar{a}_{i,s} = \frac{a_{i,s}}{a_{r,s}}\ , \quad \bar{a}_{r,j} = -\frac{a_{r,j}}{a_{r,s}}$$

where in the above formulae $i \neq r,\ j \neq s$, and the barred coefficients are the new dictionary entries computed. Using rational arithmetic, about 75-85% of the total running time was spent in the gcd computation.

An alternative to pivoting using rationals is the 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 are stored, with respect to a common denominator, which is the determinant of the current basis, denoted $det(B)$. To pivot an integer dictionary on row $r$ and column $s$ the following operations are performed:

$$\bar{a}_{i,j} = \frac{a_{i,j}a_{r,s} - a_{i,s}a_{r,j}}{det(B)} \tag{3.2}$$

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

where in the above formulae, $i \neq r$ and $j \neq s$. It can be shown that the integer division (3.2) has no remainder. Both *qrs* and *lrs* implement integer pivoting. The only gcd computations required are those to produce output coefficients in reduced form. To compare integer and rational arithmetic, we compared *ve*06 and *qrs*, see Table 6. The only essential difference between the codes is the use of integer arithmetic, so almost all of the speedup can be attributed to this. In the table, the reverse search trees for *cyclic25_13* and *metric80_16* were truncated at depth 5 to reduce the computation time. Speedups were obtained on all problems, but varied widely from 1.3 to 18.5. Columns 4 and 5 give the approximate maximum number of decimal digits used in the

| Problem | seconds | | maximum digits | | rational/integer | |
|---|---|---|---|---|---|---|
| | rational | integer | rational | integer | time | digits |
| cut16_11 | 3.8 | 1.4 | 16 | 20 | 2.7 | 0.8 |
| cut32_16 | 5749.0 | 2558.4 | 16 | 28 | 2.5 | 0.6 |
| cyclic25_13 | 4683.4 | 253.5 | 164 | 168 | 18.5 | 1.0 |
| kq20_11 | 173.0 | 11.5 | 68 | 52 | 15.0 | 1.3 |
| metric40_11 | 177.6 | 50.3 | 20 | 16 | 3.5 | 1.3 |
| metric80_16 | 554.6 | 155.8 | 16 | 16 | 3.6 | 1.0 |
| mit31_20 | 4590.2 | 3650.3 | 24 | 44 | 1.3 | 0.5 |

**Table 6: Rational vs Integer Arithmetic (** *ve06 vs qrs* )

computations. The experiments were performed using a 32-bit machine, so the number of digits reported is 4 times the number of array elements required to hold the largest number. Therefore the maximum integer may actually be up to 3 decimal digits shorter than that specified. Note that integer arithmetic appears to produce the greatest speedups for the problems involving the largest numbers.

## 3.4. Extended vs Fixed Precision

The reverse search algorithm is by nature very sensitive to numerical error. A single misinterpreted sign may mean that an entire subtree of the reverse search tree is not discovered. For this reason all implementations were performed in extended precision exact arithmetic. It is of interest to see how much overhead is involved with this arithmetic. To test this we used a version of *lrs* prepared by Marzetta called *lrs*1 which uses fixed 64 bit integers. This program contains no overflow checking, and can handle problems for which the integers in the calculations do not exceed about 19 decimal digits. Table 7 gives a comparison of *lrs*1 and *lrs*. It can be seen that the program runs about six times faster on the average when using fixed integer arithmetic. Since the overhead grows as the size of the integers grows, this can be viewed as a lower bound on the overhead of extended precision arithmetic, as implemented in *lrs*. The table indicates the importance of extending the range of problems solvable in fixed precision. The most time consuming operation is the pivot update in equation (3.2). As mentioned the division is exact. Therefore it would be of interest to derive a special purpose code to compute the value $\bar{a}_{i,j}$ without overflow, whenever it and all the terms on the right hand side of (3.2) are small enough to fit in a single computer word.

| Problem | *lrs* seconds | *lrs1* seconds | *lrs/lrs1* |
|---|---|---|---|
| cut16_11 | 0.27 | 0.06 | 4.2 |
| cut32_16 | 214.29 | 33.88 | 6.3 |
| metric40_11 | 28.11 | 4.73 | 5.9 |
| metric80_16* | 15283.01 | 2526.01 | 6.0 |

**Table 7: Extended vs Fixed Precision (** *lrs vs lrs1, * = dual polytope* )

## 3.5. Estimates

The output size of a vertex or facet enumeration problem varies enormously depending on the problem. For example, a non-redundant $V$-description containing $n$ vertices in $d$ dimensions can define a polytope with as few as $O(\log n)$ or as many as $\Omega(n^{\lfloor d/2 \rfloor})$ facets. Tight bounds are known as the Lower and Upper Bound Theorems (see, e.g. [23] ). Clearly these problems become infeasible for certain polytopes, even for rather small values of $n$ and $d$. A useful feature of reverse search is that it allows a randomization that produces unbiased estimates of the output size and the size of the reverse search tree. For polytopes given by a $V$-description, it is also possible to get an unbiased estimate of the volume.

The estimate is based on a technique of Hall and Knuth[17] and is described in Avis and Devroye[5]. The key fact is that it is possible at any node in a reverse search tree to find its degree and generate one of its children uniformly at random. This allows the generation of a random path from the root to a leaf. An estimate of the tree size is calculated from the degree sequence obtained. The estimate is unbiased, meaning that its expected value is the correct number of nodes in the tree. It is also possible to get an estimate of the expected value of any function defined at each node of the tree. For a vertex enumeration problem, one such function is the indicator function, with value one if the corresponding basis is lexmin for its vertex, and zero otherwise. The expected value of this function is the number of vertices of the polyhedron. Similary we can determine the number of extreme rays by letting the function be the number of lexmin rays defined by the dictionary with the current basis. For facet enumeration problems, we can compute the expected value of the volume by letting the function represent the determinant of the current basis, and scaling appropriately at the end of the computation.

Although the estimates obtained are unbiased, they have high variance. Methods for reducing the variance are described in[5]. The method implemented in *lrs* is to search the tree completely to a fixed depth *h*, and then to make random probes to estimate the size of each remaining subtree. Table 8 gives the results for a set of estimates for our test problems. The parameter *h* is shown in the final column. For the first problem, 3 random probes were made from the root and averaged. The other estimates were obtained by a single random probe in each subtree remaining at the depth *h*. Column 2 gives the upper bound on the number of vertices or facets given by the Upper Bound Theorem applied to problems of each given size. This also gives an upper bound on the number of bases that will be generated by *lrs*, since these bases correspond to the bases of a perturbed polyhedron with the same input parameters as the original problem. In most cases this is a huge overestimate. For *cyclic*25_13 however it is exact, since the class of cyclic polytopes realize the bound given by the Upper Bound Theorem. The next three columns give estimates for the number of vertices (or facets), lex positive bases(#B) and the volume. These can be compared with the actual values given in Table 3. Since the running time of *lrs* is directly proportional to #B, the estimate of this parameter provides an easy way to estimate the running time required to solve the problem competely. The ratio of the estimate to the actual value is given in the next three columns. The second last column shows the percentage of the entire reverse search tree that was examined in making the estimate.

| Problem | Upper | Estimate | | | Estimate/Actual | | | Eval | *h* |
|---|---|---|---|---|---|---|---|---|---|
| | Bound | #V(#F) | #B | Volume | #V(#F) | #B | Vol. | % | |
| cut16_11 | 672 | 51 | 58 | 1.04 | 0.91 | 0.47 | 0.45 | 5.6 | 0 |
| cut32_16 | 692208 | 648 | 29766 | 0.86 | 1.76 | 0.63 | 0.64 | 4.2 | 5 |
| cyclic25_13 | 35700 | 24454 | 11082 | 3.16e+74 | 0.68 | 0.60 | 0.07 | 6.5 | 4 |
| kq20_11 | 4004 | 761 | 761 | - | 0.64 | 0.64 | - | 2.8 | 1 |
| metric40_11 | 371008 | 20 | 6929 | - | 0.63 | 0.75 | - | 8.4 | 4 |
| metric80_16 | 2946219408 | 229 | 3800960 | - | 0.42 | 0.58 | - | 1.0 | 6 |
| mit31_20 | 587860 | 6096 | 18549 | 8.216e+2 | 0.32 | 0.51 | 0.48 | 4.5 | 3 |

**Table 8: Estimates(***lrs***)**

For problems where the determinants of the bases have roughly the same absolute value, the error in estimating the volume will be comparable to that in estimating the number of bases. For the cyclic polytope the determinants of the bases have enormous variance, explaining the relatively poor estimate obtained in this case.

## 4. Conclusion

In this paper we gave empirical evidence that symbolic perturbation, lifting and integer pivoting all give substantial speedups when used in a reverse search vertex enumeration program. These methods are all quite general and suitable for other geometric and polyhedral computation problems. We also gave some indication of the overhead cost of extended precision arithmetic and demonstrated *lrs*'s estimation feature.

Although *lrs* is a large improvement on earlier implementations, 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 [7] 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 such as the cut and metric polytopes described in the paper. For these polytopes a double description algorithm, such as Fukuda's *cdd* [2] is superior. *lrs* is efficient for vertex enumeration of simple (or near-simple) polyhedra, or dually for facet enumeration of simplicial (or near-simplicial) polyhedra. It is also useful when the output size is too large to be stored in memory. 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 [11].

*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.

## 5. Acknowledgements

The author would like to thank Masao Iri for his interest in and encouragement of this project from the outset, and for teaching him the importance of empirical work. This paper was written while the author was on leave visiting the group of Tom Liebling at EPFL in Lausanne, whose support is gratefully acknowledged. An anonymous referee suggested the inclusion of Table 1. 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!

## 6. Appendix

The program *lrs*, and the input files for the polyhedra described in this paper can be obtained by accessing the *lrs* home page[1]. Version 3.2 was used for the tests described in this paper, except for Table 8 which was produced by Version 3.2b, the first version which produces volume estimates. The implementations *ve*01, *ve*06 and *qrs*, Version 2.2a are available from the author on request. The computational results in Tables 3, 4 and 5 were obtained on *masg*40, an SGI Origin 200 running Irix 6.4, and those in Table 6 on *masg*20, an SGI O2 running Irix 6.3, both at EPFL. The results in Tables 2, 7 and 8 were obtained by *mutt*, a DEC AlphaServer 1000 4/23 at McGill.

## References

1. *lrs Home Page*, November 1997. ftp://mutt.cs.mcgill.ca /pub/C/lrs.html
2. *cdd+ Homepage*, December, 1997. http://www.ifor.math.ethz.ch/staff/fukuda/cdd_home/cddman.html
3. D. Avis, ''On the Extreme Rays of the Metric Cone,'' *Canadian J. of Maths.*, vol. 32, pp. 126-144, 1980.
4. 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.
5. 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

6. 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
7. D. Avis, D. Bremner, and R. Seidel, ‘‘How Good are Convex Hull Algorithms?,’’ *Computational Geometry: Theory and Applications*, vol. 7, pp. 265-301, 1997.
8. D. Avis, *lrs*: A Revised Implementation of the Reverse Search Vertex Enumeration Algorithm, May 1998. ftp://mutt.cs.mcgill.ca /pub/doc/avis/Av98a.ps.gz
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. M. Bussieck and M. Luebbecke, ‘‘The Vertex Set of a 0/1-Polytope is Strongly P-Enumerable,’’ *ISMP ’97*, Lausanne, August 1997.
12. G. Ceder, G.D. Garbulsky, D. Avis, and K. Fukuda, ‘‘Ground States of a Ternary Lattice Model with Nearest and Next-Nearest Neighbor Interactions,’’ *Physical Review B*, vol. 49, pp. 1-7, 1994.
13. V. Chvátal, *Linear Programming*, W.H. Freeman, 1983.
14. M. Deza and M. Laurent, *Geometry of Cuts and Metrics*, Springer, 1997.
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. M. Iri, ‘‘On an Extension of the Maximum-flow Minimum-cut Theorem to Multicommodity Flows,’’ *J. Oper. Soc. Japan*, vol. 13, pp. 129-135, 1970/71.
20. D. E. Knuth, *The Art of Computer Programming, Vol 2: Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1981.
21. H.W. Kuhn and R.E. Quandt, ‘‘An Experimental Study of the Simplex Method,’’ *Proc. of Symposia in Applied Mathematics*, vol. 15, pp. 107-124, 1963.
22. M. Lomonosov, ‘‘Combinatorial Approaches to Multiflow Problems,’’ *Discrete Applied Math.*, vol. 11, pp. 1-93, 1985.
23. G. Ziegler, *Lectures on Polytopes*, Springer, 1994, revised 1998. Graduate Texts in Mathematics.