Reverse Search for Enumeration

(Research Report No. 92-5)

by

David Avis
and
Komei Fukuda

April 24, 1992

Graduate School of Systems Management
The University of Tsukuba
3-29-1 Otsuka, Bunkyo-ku
Tokyo 112, Japan

A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra

David Avis
School of Computer Science
McGill University
Montréal, Canada

Komei Fukuda
Graduate School of Systems Management
The University of Tsukuba
Tokyo, Japan

Abstract:

We present a new pivot-based algorithm which can be used with minor modification for the enumeration of the facets of the convex hull of a set of points, or for the enumeration of the vertices of an arrangement or of a convex polyhedron, in arbitrary dimension. The algorithm has the following properties:

(a) No additional storage is required beyond the input data;
(b) The output list produced is free of duplicates;
(c) The algorithm is extremely simple, requires no data structures, and handles all degenerate cases;
(d) The running time is output sensitive for non-degenerate inputs;
(e) The algorithm is easy to efficiently parallelize.

For example, the algorithm finds the v vertices of a polyhedron in R^d defined by a non-degenerate system of n inequalities (or dually, the v facets of the convex hull of n points in R^d, where each facet contains exactly d given points) in time $O(ndv)$ and $O(nd)$ space. The v vertices in a simple arrangement of n hyperplanes in R^d can be found in $O(n^2dv)$ time and $O(nd)$ space complexity. The algorithm is based on inverting finite pivot algorithms for linear programming.

1. Introduction

In this paper we give an algorithm, which with minor variations can be used to solve three basic enumeration problems in computational geometry: facets of the convex hull of a set of points, vertices of a convex polyhedron given by a system of linear inequalities, and vertices of an arrangement of hyperplanes. The algorithm is based on "inverting" finite pivoting algorithms for linear programming. For terms not defined here, the reader is referred to Chvátal[4] for linear programming and Edelsbrunner[7] for arrangements. In the the rest of this section we give an informal description of the algorithm beginning with the vertex enumeration problem for convex polyhedra.

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.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1991 ACM 0-89791-426-0/91/0006/0098 $1.50

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 [7]). From the vertices of the Voronoi diagram, the cells of the Delaunay triangulation of the point set can be readily obtained with no additional computation. A variant of our method can be used for vertex enumeration of arrangements.

The problems discussed in this paper have a long history, which we briefly mention here. The problem of enumerating all of the vertices of a polyhedron is surveyed by Mattheiss and Rubin in[12] and by Dyer in[5]. There are essentially two classes of methods. One class is based on pivoting and is discussed in detail in[5] and[4]. In this method a depth first search is initiated from a vertex by trying all possible simplex pivots. The difficulty is in determining whether or not a vertex has already been visited. For this all vertices must be stored in a balanced AVL-tree. An implementation that takes $O(nd^2v)$ time and $O(dv)$ space for a polyhedron with v vertices defined by a non-degenerate system of n inequalities in R^d is given in[5]. A dual version that computes convex hulls was discovered by Chand and Kapur[3], and has similar complexity. Using sophisticated data structures, Seidel[14] was able to achieve a running time of $O(d^3 v \log n + nf(d-1, n-1))$ for sets of n points in general position in R^d. Here $f(d, n)$ is the time to solve a linear program with n constraints in d variables, and v is the number of facets of the convex hull. The space required for this algorithm is $O(n^{\lfloor d/2 \rfloor})$. The algorithm presented in this paper fits into this class. It achieves $O(dvn)$ time and $O(dn)$ space complexity for facet enumeration of the convex hull of n points in R^d, when each facet contains exactly d given points. To the authors' knowledge, it is the only algorithm known that has non-exponential space requirements in the worst case.

A second class of methods for computing the vertices of a convex polyhedron is the "double description" method of Motzkin et al.[13] that dates back to 1953. In fact the origin of these methods is even earlier, as the double description method is in fact dual to the Fourier-Motzkin method for the solution of linear inequality systems. In the double description method, the polyhedron is constructed sequentially by adding a constraint at a time. All new vertices produced must lie on the hyperplane bounding the constraint currently being inserted. A dual version for constructing convex hulls is known in the computational geometry community as the "beneath and beyond" method. Assuming the dimension d is fixed, the fastest algorithm of this class again uses sophisticated data structures and is due to Seidel [15] (also see [7]). It takes $O(n^{\lfloor d/2+1 \rfloor})$ time and $O(n^{\lfloor d/2 \rfloor})$ space.

With d fixed, the complete facial structure of a hyperplane arrangement can be constructed by an algorithm due to Edelsbrunner, O'Rourke and Seidel [6] in optimal time and space $O(n^d)$. The algorithm works by inserting the hyperplanes one at a time and can handle degenerate cases. Again with d fixed, a method for enumerating just the edges and vertices (with repetitions) in $O(n^d)$ time and $O(n)$ space is given by Edelsbrunner and Guibas[8]. Houle et al.[11] give several applications in data approximation where it is required to enumerate all vertices of an arrangement.

2. Dictionaries

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

$$A\,x = 0, \quad x_g = 1. \tag{2.1}$$

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 (2.1) 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 (2.1) that have at least one basis, and will assume this for the rest of the paper. Given any basis B, we can transform (2.1) into the *dictionary*:

$$x_B = -A_B^{-1}A_N \, x_N = \bar{A} \, x_N, \qquad (2.2)$$

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. 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*. In section 2 of the paper we give an algorithm for enumerating all distinct basic solutions of the system (2.1) without repetition, using only the space required to store the input. The algorithm is initiated with an optimal dictionary. A variant of the algorithm enumerates all primal feasible dictionaries reporting the corresponding basic feasible solutions without repetition.

The geometric problems mentioned in the title can all be transformed to dictionary enumeration problems. A *hyperplane* in R^d, $d \geq 0$, is denoted by the pair (b, c), where b is a vector of length d and c is a scalar, and is the solution set of the equation $by = c$, $y = (y_j : j = 1, ..., d)$. A *hyperplane arrangement* is a collection of n_0 hyperplanes (b_i, c_i), for some integer n_0. A *vertex* of the arrangement is the unique solution to the system of d equations corresponding to d intersecting hyperplanes. The *vertex enumeration problem* for hyperplane arrangements is to list all of the vertices of an arrangement. From a hyperplane arrangement, we show how to construct an optimal dictionary with $m = n_0-d+1$ and $n = n_0+2$.

A (convex) polyhedron P is the solution set to a system of n_0 inequalities in d non-negative variables: $P = \{y \in R^d \mid A'y \leq b, y \geq 0\}$, 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. In fact to find even a single vertex of P is computationally equivalent to linear programming. As we wish to separate this from the enumeration problem, we will assume we are given an initial vertex. We show how to construct an optimal dictionary for P with $m = n_0+1$ and $n = n_0+d+2$. It can be shown that each *primal feasible* dictionary has a basic solution which gives a vertex y of P.

Let $Q = \{q_1, ..., q_{n_0}\}$ denote a set of n_0 points in R^d. A *facet* of the convex hull of Q is a hyperplane containing d affinely independent points of Q. There is no loss of generality in assuming that the origin is contained in the convex hull of Q. By employing a standard duality between points and hyperplanes, we may transform the *facet enumeration* problem for Q into a vertex enumeration problem for a convex polyhedron.

3. Enumeration of Dictionaries

Suppose we are given a system of equations of the form (2.1), for some $m \times n$ matrix A. The *linear programming problem* (LP) for (2.1) is to maximize x_f over (2.1) subject to the additional constraint that each variable except x_f and x_g is non-negative. Each optimal dictionary is a solution to LP. To begin with, we will assume that there is a unique optimal dictionary. 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

$$a'_{rs} = -\frac{1}{\bar{a}_{rs}}, \quad a'_{is} = \frac{\bar{a}_{is}}{\bar{a}_{rs}}, \quad a'_{rj} = -\frac{\bar{a}_{rj}}{\bar{a}_{rs}}, \qquad (3.1)$$

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

The pivot is *primal feasible* (respectively, *dual feasible*) if both of the dictionaries corresponding to B and B' are primal (respectively, dual) feasible. The simplex method is a method of solving LP by beginning with an initial dictionary and pivoting until an optimal dictionary is found. We consider two rules for choosing a pivot. The first rule, known as

Bland's rule starts with primal feasible basis B.

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. The second rule, known as the criss-cross rule, starts with any basis.

Criss-Cross Rule

(1) Let $i \neq f, g$ be the smallest index such that x_i is (primal or dual)infeasible.

(2) If $i \in B$, let $r=i$ and let s be the minimum index such that $\bar{a}_{rs} > 0$, otherwise let $s=i$ and let r be the minimum index such that $\bar{a}_{rs} < 0$.

The criss-cross pivot (r, s) interchanges x_r and x_s, and may not preserve either primal or dual feasibility. In both cases, if step (1) does not apply then the dictionary is optimal.

The validity of these rules is given by the following proposition. Part (a) is proved in[2] and part (b) in[16] for linear programs, and in[17] [19] in the more general setting of oriented matroids. A simple proof of part (b) also appears in [10].

Proposition 3.1. Let (2.1) be a system that admits an optimal dictionary and let B be any basis.
(a) If B is primal feasible, then successive application of Bland's rule leads to an optimal dictionary, and each basis generated is primal feasible.
(b) Successive application of the criss-cross rule starting with basis B leads to an optimal dictionary.

First we give a dictionary enumeration algorithm for systems (2.1) that admit a unique optimal dictionary. Consider a graph where vertices are dictionaries and two vertices are adjacent if the corresponding two dictionaries differ in only one basic variable. Then part (b) of the proposition tells us that there is a unique path consisting of criss-cross pivots from any dictionary to the optimal dictionary. The set of all such paths gives us a spanning tree in this graph. Consider a non-optimal dictionary D with basis B. Let (r, s), $r \in B-f$, $s \in N-g$, be the pivot obtained by applying the criss-cross rule to D giving a dictionary D'. We call (s, r) a *reverse criss-cross pivot* for D'. Suppose we start at the optimal dictionary and explore reverse criss-cross pivots in lexicographic order. This corresponds to a depth first search of the spanning tree defined above. When moving down the tree, each dictionary is encountered exactly once. A similar analysis applies to part (a) of the proposition. We form a similar graph, except that vertices are just the primal feasible dictionaries. We define a *reverse Bland pivot* in the analogous way. A depth first search of this graph provides all primal feasible dictionaries.

Our enumeration algorithm *search* for dictionaries is given in Figure 3.1. For a given system (2.1) we have an initial basis $B = \{1, ..., m\}$, co-basis $N = \{m+1, ..., n\}$ and optimal dictionary $x_B = \bar{A}x_N$. We further assume that $f = 1$, $g = n$, and that m and n are global constants. The efficiency of the procedure depends greatly on the procedure *reverse*. The simplest way to check if (r, s), $r \in B-f$, $s \in N-g$, is a reverse pivot is to actually perform the pivot, then use procedure *select-pivot* on the new dictionary. If this produces the same pair of variables, then (r, s) is a valid reverse pivot. Since a pivot involves $O(mn)$ operations, a faster method is desirable. In fact to determine the pivot by the criss-cross or Bland's rules, the entire dictionary is not required by procedure *select-pivot*. To test whether $\bar{A}$ arises from a coefficient matrix A' by a criss-cross (resp., Bland) pivot interchanging $B[i]$ with $N[j]$, it is only necessary to examine rows f, i and columns j, g of A'. These can be computed from $\bar{A}$ in $O(m+n)$ time, and checked to see if $(B[i], N[j])$ is a criss-cross (resp., Bland) pivot. Further savings are possible, as certain potential reverse pivots can be eliminated without any pivoting.

Proposition 3.2 If (s, r), $s \in B-f$ and $r \in N-g$, is a valid reverse criss-cross pivot for a dictionary $x_B = \bar{A}x_N$, then either

(a) $\bar{a}_{sg} > 0$, $\bar{a}_{sr} > 0$, $\bar{a}_{sj} \geq 0$ for $j \in N-g, j < s$, or

(b) $\bar{a}_{fr} < 0$, $\bar{a}_{sr} < 0$, $\bar{a}_{ir} \leq 0$ for $i \in B-f, i < r$.

Proposition 3.3 Let $x_B = \bar{A}x_N$ be a dictionary, let $r \in N-g$ and set $\lambda = \min\{-\frac{\bar{a}_{ig}}{\bar{a}_{ir}} : i \in B-f, \bar{a}_{ir} < 0\}$. If (s, r), $s \in B-f$, is a valid reverse Bland's rule pivot then s must be an index that obtains this minimum.

```
procedure search $(B, N, \bar{A})$;
/* $B = \{1, ..., m\}$, $N = \{m+1, ..., n\}$, $f = 1$, $g = n$,
$x_B = \bar{A}x_N$ is a unique optimal dictionary for a system
(2.1) */
  begin
    $i$:=2; $j$:=1;
    repeat
      while ( $i \le m$ and not reverse
      $(B, N, \bar{A}, i, j)$ ) increment $(i, j)$;
      if ( $i \le m$ ) then      /* reverse pivot
      found */
        begin
          pivot $(B, N, \bar{A}, i, j)$;
          if lex-min $(B, N, \bar{A})$ then print $(B)$;
          $i$:=2; $j$:=1;
        end;
      else       /* go back to previous diction-
      ary */
        begin
          select-pivot $(\bar{A}, i, j)$;
          pivot $(B, N, \bar{A}, i, j)$;
          increment $(i, j)$;
        end;
    until ($i > m$ and $B[m] = m$ )
  end;      /* search */

function reverse $(B, N, \bar{A}, i, j)$:boolean;
/* true if $(s, r)$, with $s = B[i]$, $r = N[j]$, is a
valid reverse cross-pivot (resp., Bland-pivot ) for
$\bar{A}$, otherwise false */

procedure pivot $(B, N, \bar{A}, i, j)$;
/* pivot $\bar{A}$ on row $i$ and column $j$, update $B$ and
$N$. Reorder as necessary and set $i$ and $j$ to be the
indices of the interchanged $B[i]$ and $N[j]$. */

function lex-min $(B, N, \bar{A})$:boolean;
/* true if $\bar{A}$ is non-degenerate, or degenerate and
$B$ is the lexicographically minimum basis for this
basic solution, else false */

procedure select-pivot $(\bar{A}, i, j)$;
/* Find criss-cross (resp., Bland) pivot for
coefficient matrix $\bar{A}$. Return the index $i$ of the
pivot row and index $j$ of the pivot column*/

procedure increment $(i, j)$;
  begin
    $j$:=$j$+1;  if  ($j = n-m$)  then  begin
    $j$:=1; $i$:=$i$+1; end;
  end;      /* increment */
```

Figure 3.1

The procedure *lex−min* is used to ensure that each basic solution is output exactly once, when the lexicographically minimum basis for that basic solution is reached. The correctness of the procedure is based on the following proposition.

Proposition 3.4 Let B be a basis for a degenerate dictionary $x_B = \bar{A}x_N$. B is *not* lexicographically minimum for the corresponding basic solution if and only if there exists $r \in B-f$ and $s \in N-g$ such that $r > s$, $\bar{a}_{rg} = 0$ and $\bar{a}_{rs} \ne 0$. □

Procedure *search* as given in the previous subsection will only generate all (feasible) dictionaries if the system (2.1) has a unique optimal dictionary. Suppose there are many optimal dictionaries. This situation arises when one of the basic variables has value zero, ie. the dictionary is degenerate. Then instead of a spanning tree in the graph described after Proposition 3.1, we obtain a spanning forest. Each of the two pivot algorithms terminates when any optimal solution is found. Therefore, procedure *search* must be applied to each optimal dictionary. Fortunately, from any optimal dictionary we can generate all optimal dictionaries by a procedure very similar to search. We construct a non-degenerate optimum dictionary by adding an extra column, and use the dual form of Bland's rule to maintain dual feasibility while constructing all optimal dictionaries to the original problem. Details are given in the full paper[1].

4. Complexity

In this section we discuss the complexity of the dictionary enumeration algorithm, and apply the results to the geometric applications described in Section 2. Suppose we have a system (2.1) for some $m \times n$ matrix A. Let $f(A)$ denote the number of dictionaries that can represent (2.1). $f(A)$ is just the number of linearly independent subsets of m columns of A, with the condition that the column with index f is always included, and index g is always excluded. This is at most $\binom{n-2}{m-1}$, but may be much smaller. For each dictionary, we may evaluate $(m-1)(n-m-2)$ candidates for reverse pivots, each candidate requiring $O(m+n)$ time as shown in the previous section. Procedure *pivot* requires $O(m(n-m))$ time per execution as does procedure *lex−min*. These complexities are valid for the case of multiple optimal solutions. Therefore the overall time-complexity of *search* is

$$O((m+n)m(n-m)f(A)) = O\left((m+n)mn \binom{n-2}{m-1}\right) \quad (4.1)$$

Apart from a few indices, no additional space is required other than that required to represent the input.

We now consider the complexity of evaluating all feasible dictionaries. Let $g(A)$ denote the number of primal feasible dictionaries representing (2.1). The above analysis and (4.1) hold, with $g(A)$ replacing $f(A)$. In the non-degenerate case we can do better. Recalling Proposition 3.3, we see that we only need to consider one candidate reverse pivot per column of the dictionary: if there are two or more indices realizing the minimum then a pivot would give a degenerate dictionary. For each column, the candidate basic variable can be found by computing the minimum ratio λ in $O(m)$ time. To check if a candidate is in fact a reverse pivot, we need to construct the objective row of the dictionary after the pivot, taking $O(n-m)$ time. Therefore since there are $n-m-2$ candidate columns, all reverse Bland pivots from the given dictionary can be found in $O((n-m)n)$ time, in the non-degenerate case. This gives an overall complexity of $O((n-m)ng(A))$ for the non-degenerate case.

We now return to the geometric problems mentioned in Section 2. Suppose we have a collection of n_0 hyperplanes in R^d. For this problem, $m = n_0-d+1$ and $n = n_0+2$. The time-complexity of enumerating all vertices of a hyperplane arrangement by this method becomes: $O(n_0^2\, d\, f(A))$. In the case of non-degenerate arrangements, $f(A)$ is the number of the vertices, ie. the size of the output. This method should be particularly useful for non-degenerate arrangements with few vertices.

Consider now the enumeration of the vertices of a polyhedron given by a list of n_0 inequalities in d variables. Since we assume the polyhedron has at least one vertex, $n_0 \geq d$. We have $m = n_0+1$ and $n = n_0+d+2$. The time-complexity of enumerating all of the vertices is $O(n_0^2 d\; g(A))$. Again the complexity is output sensitive for non-degenerate polyhedra, for which $g(A)$ is just the number of vertices. If the polyhedron is simple (ie. all dictionaries are non-degenerate) then we get an improved complexity bound. The algorithm produces vertices at a cost of $O(n_0 d)$ per vertex with no repetitions and no additional space. These complexities apply to the convex hull problem, where n_0 is the number of input points. In the non-degenerate case where no more than d points lie on any facet (ie., the facets are simplicial), we can enumerate the v facets in time $O(n_0 dv)$ and space $O(n_0 d)$.

The simplicity of the algorithm rends it suitable for symbolic computation in a language such as *Maple* or *Mathematica*. Using exact arithmetic, the problem of numerical accuracy which occurs with most geometric algorithms is avoided. Another feature of the algorithm is that it is easy to efficiently parallelize. Since in the enumeration no dictionary is ever reached by two different paths and no additional storage is required, subproblems can be scheduled arbitrarily onto free processors.

The complexity analysis presented for the degenerate case is quite rudimentary. We allow a worst case time of $O(m+n)$ to determine whether a pair of indices is a reverse pivot. This seems certain to be an overestimate. In a reverse criss-cross pivot, for the ith basic variable to interchange with the jth non-basic variable, at least $i+j$ signs have to be "correct". We may compute these signs consecutively and stop the first time an "incorrect" sign is encountered. Amortizing this cost over the complete enumeration of an arrangement, it is possible that just a constant amount of work has to be done on the average to determine that a potential reverse pivot is invalid.

The "reverse pivoting" approach can be extended to the setting of oriented matroids, and in particular to pseudo line arrangements. While the criss-cross method works correctly in the setting of oriented matroids, Bland's rule is not finite for oriented matroid programming [9]. Todd[18] has found a finite rule that can replace Bland's rule in the oriented matroid setting.

5. Acknowledgements

The work of the first author was performed while visiting the laboratory of Professor Masakazu Kojima of Tokyo Institute of Technology, supported by the JSPS/NSERC bilateral exchange program.

References

1. D. Avis and K. Fukuda, "A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra," *Technical Report B-237*, Tokyo Institute of Technology, Dept. of Information Science, November 1990.
2. R. G. Bland, "A Combinatorial Abstraction of Linear Programming," *J. Combin. Theory B*, vol. 23, pp. 33-57, 1977.
3. D.R. Chand and S.S. Kapur, "An Algorithm for Convex Polytopes," *J. ACM*, vol. 17, pp. 78-86, 1970.
4. V. Chvátal, *Linear Programming*, W.H. Freeman, 1983.
5. M.E. Dyer, "The Complexity of Vertex Enumeration Methods," *Math. Oper. Res.*, vol. 8, pp. 381-402, 1983.
6. H. Edelsbrunner, J. O'Rourke, and R. Seidel, "Constructing Arrangements of Lines and Hyperplanes with Applications," *SIAM J. Computer Science*, pp. 341-363, 1986.
7. H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1987.
8. H. Edelsbrunner and L. Guibas, "Topologically Sweeping an Arrangement," *J. Comp. Syst. Sciences*, vol. 38, pp. 165-194, 1989.
9. K. Fukuda, "Oriented Matroid Programming," *Ph.D. Thesis*, University of Waterloo, 1982.
10. K. Fukuda and T. Matsui, "On the Finiteness of the Criss-Cross Method," *European J. O.R.*, to appear.
11. M. E. Houle, H. Imai, K. Imai, J-M. Robert, and P. Yamamoto, "Orthogonal Weighted Linear L_1 and L_∞ Approximation and Applications," *manuscript*, September 1990.
12. T.H. Matheiss and D. S. Rubin, "A Survey and Comparison of Methods for Finding all Vertices of Convex Polyhedral Sets," *Math. Oper. Res.*, vol. 5, pp. 167-185, 1980.
13. T.S. Motzkin, H. Raiffa, G.L. Thompson, and R. M. Thrall, "The Double Description Method," *Annals of Math. Studies 8*, Princeton University Press, 1953.
14. R. Seidel, "Constructing Higher-Dimensional Convex Hulls at Logarithmic Cost per Face," *Proc. 1986 S.T.O.C.*, pp. 404-413.
15. R. Seidel, "A Convex Hull Algorithm Optimal for Point Sets in Even Dimensions," *Report 81-14*, University of British Columbia, Dept. of Computer Science, 1981.
16. T. Terlaky, "A Convergent Criss-Cross Method," *Math. Oper. und Stat. ser. Optimization*, vol. 16, pp. 683-690, 1985.
17. T. Terlaky, "A Finite Criss-Cross Method for Oriented Matroids," *J. Combin. Theory B*, vol. 42, pp. 319-327, 1987.
18. M. Todd, "Linear and Quadratic Programming in Oriented Matroids," *J. Comb. Theory B*, vol. 39, pp. 105-133, 1985.
19. Z. Wang, "A Conformal Elimination Free Algorithm for Oriented Matroid Programming," *Chinese Annals of Mathematics*, vol. 8,B,1, 1987.