cddlib Reference Manual

Komei Fukuda
School of Computer Science
McConnel Engineering Building
3480 University Street
Montreal, Quebec
Canada H3A 2A7

email: fukuda@cs.mcgill.ca

(cddlib ver. 092, manual ver. December 19, 2001)

Abstract

This is a reference manual for cddlib-092. The manual is very far from satisfactory but explains the most important functions of polyhedral representation conversion in cddlib. Please use the accompanying README file and test programs to complement the incompleteness.

1 Introduction

The program cddlib is an efficient implementation [12] of the double description Method [14] for generating all vertices (i.e. extreme points) and extreme rays of a general convex polyhedron given by a system of linear inequalities:

$$P = \{x = (x_1, x_2, \ldots, x_d)^T \in R^d : b - Ax \geq 0\}$$

where A is a given $m \times d$ real matrix and b is a given real m-vector. In the mathematical language, the computation is the transformation of an *H-representation* of a convex polytope to an *V-representation*.

cddlib is a C-library version of the previously released C-code cdd/cdd+. In order to make this library version, a large part of the cdd source (Version 0.61) has been rewritten. This library version is more flexible since it can be called from other C programs.

One useful feature of cddlib/cdd/cdd+ is its capability of handling the dual (reverse) problem without any transformation of data. The dual transformation problem of a V-representation to a minimal H-representation and is often called the *(convex) hull problem*. More explicitly, is to obtain a linear inequality representation of a convex polyhedron given as the Minkowski sum of the convex hull of a finite set of points and the nonnegative hull of a finite set of points in R^{d+1}:

$$P = conv(v_1, \ldots, v_n) + nonneg(r_{n+1}, \ldots, r_{n+s}),$$

where the *Minkowski sum of two subsets S and T* of R^{d+1} is defined as

$$S + T = \{s + t \,|s \in S \text{ and } t \in T\}.$$

As we see in this manual, the computation can be done in straightforward manner. Unlike the earlier versions of cdd/cdd+ that assume certain regularity conditions for input, cddlib is designed to do a correction transformation for any general input. The user must be aware of the fact that in certain cases the transformation is not unique and there are polyhedra with infinitely many representations. For example, a line segment (1-dimensional polytope) in R^3 has infinitely many minimal H-representations, and a halfspace in the same space has infinitely many minimal V-representations. cddlib generates merely one minimal representation.

cddlib comes with an LP code to solve the general linear programming (LP) problem to maximize (or minimize) a linear function over polyhedron P. It is useful mainly for solving dense LP's with large m (say, up to few hundred thousands) and small d (say, up to 100). It implements a revised dual simplex method that updates $(d+1) \times (d+1)$ matrix for a pivot operation.

The program cddlib has an I/O routines that read and write files in *Polyhedra format* which was defined by David Avis and the author in 1993, and has been updated in 1997. The program called lrs [2] developed by David Avis is a C-implementation of the reverse search algorithm [4] for the same enumeration purpose, and it conforms to Polyhedra format as well. Hopefully, this compatibility of the two programs enables users to use both programs for the same input files and to choose whichever is useful for their purposes. From our experiences with relatively large problems, the two methods are both useful and perhaps complementary to each other. In general, the program cdd+ tends to be efficient for highly degenerate inputs and the program rs tends to be efficient for nondegenerate or slightly degenerate problems.

Although the program can be used for nondegenerate inputs, it might not be very efficient. For nondegenerate inputs, other available programs, such as the reverse search code lrs or qhull (developed by the Geometry Center), might be more efficient. See Section 8 for pointers to these codes. The paper [3] contains many interesting results on polyhedral computation and experimental results on cdd+, lrs, qhull and porta.

I will not take any responsibility of any problems you might have with this program. But I will be glad to receive bug reports or suggestions at the e-mail addresses above. Finally, if cdd+ turns out to be useful, please kindly inform me of what purposes cdd has been used for. I will be happy to include a list of applications in future distribution if I receive enough replies. The most powerful support for free software development is user's appreciation and collaboration.

2 Polyhedra H- and V-Formats (Version 1999)

Every convex polyhedron has two representations, one as the intersection of finite halfspaces and the other as Minkowski sum of the convex hull of finite points and the nonnegative hull of finite directions. These are called H-representation and V-representation, respectively.

Naturally there are two basic Polyhedra formats, H-format for H-representation and V-format for V-representation. These two formats are designed to be almost indistinguishable, and in fact, one can almost pretend one for the other. There is some asymmetry arising from the asymmetry of two representations.

First we start with the H-representation. Let A be an $m \times d$ matrix, and let b be a column m-vector. The Polyhedra format (*H-format*) of the system $b - Ax \geq \mathbf{0}$ of m inequalities in d variables $x = (x_1, x_2, \ldots, x_d)^T$ is

various comments
H-representation
(**linearity** $t\ \ i_1\ \ i_2\ \ \ldots\ \ i_t$)
begin
$m \quad d+1$ numbertype
$b \quad -A$
end
various options

where numbertype can be one of integer, rational or real. When rational type is selected, each component of b and A can be specified by the usual integer expression or by the rational expression "p/q" or "$-p/q$" where p and q are arbitrary long positive integers (see the example input file rational.ine). In the 1997 format, we introduced "H-representation" which must appear before "begin". There was one restriction in the old polyhedra format (before 1997): the last d rows must determine a vertex of P. This is obsolete now.

In the new 1999 format, we added the possibility of specifying **linearity**. This means that for H-representation, some of the input rows can be specified as **equalities**: $b_{i_j} - A_{i_j} = 0$ for all $j = 1, 2, \ldots, t$. The linearity line may be omitted if there are no equalities.

Option lines can be used to control computation of a specific program. In particular both cdd and lrs use the option lines to represent a linear objective function. See the attached LP files, samplelp*.ine.

Next we define Polyhedra *V-format*. Let P be represented by n gerating points and s generating directions (rays) as $P = conv(v_1, \ldots, v_n) + nonneg(r_{n+1}, \ldots, r_{n+s})$. Then the Polyhedra V-format for P is

various comments		
V-representation		
(**linearity** $t\ \ i_1\ \ i_2\ \ \ldots\ \ i_t$)		
begin		
$n+s$	$d+1$	numbertype
1	v_1	
$\vdots$	$\vdots$	
1	v_n	
0	r_{n+1}	
$\vdots$	$\vdots$	
0	r_{n+s}	
end		
various options		

Here we do not require that vertices and rays are listed separately; they can appear mixed in arbitrary order.

Linearity for V-representation specifies a subset of generators whose coefficients are relaxed to be **free**: for all $j = 1, 2, \ldots, t$, the $k = i_j$th generator (v_k or r_k whichever is the i_jth generator) is a free generator. This means for each such a ray r_k, the line generated by r_k is in the polyhedron, and for each such a vertex v_k, its coefficient is no longer nonnegative but still the coefficients for all v_i's must sum up to one.

When the representation statement, either "H-representation" or "V-representation", is omitted, the former "H-representation" is assumed.

It is strongly suggested to use the following rule for naming H-format files and V-format files:

(a) use the filename extension ".ine" for H-files (where ine stands for inequalities), and

(b) use the filename extension ".ext" for V-files (where ext stands for extreme points/rays).

3 Basic Object Types (Structures) in cddlib

Here are the types (defined in cddtypes.h) that are important for the cddlib user. The most important one, **dd_MatrixType**, is to store a Polyhedra data in a straightforward manner. Once the user sets up a (pointer to) dd_MatrixType data, he/she can load the data to an internal data type (dd_PolyhedraType) by using functions described in the next section, and apply the double descrition method to get another representation. As an option dd_MatrixType can save a linear objective function to be used by a linear programming solver.

```
typedef long dd_rowrange;
typedef long dd_colrange;
typedef long dd_bigrange;

typedef set_type dd_rowset;   /* set_type defined in setoper.h */
typedef set_type dd_colset;
typedef long *dd_rowindex;
typedef int *dd_rowflag;
typedef long *dd_colindex;
typedef mytype **dd_Amatrix;  /* mytype is either GMP mpq_t or 1-dim double array. */
typedef mytype *dd_Arow;

typedef enum {
  dd_Real, dd_Rational, dd_Integer, dd_Unknown
} dd_NumberType;

typedef enum {
  dd_Inequality, dd_Generator, dd_Unspecified
} dd_RepresentationType;

typedef enum {
  dd_InProgress, dd_AllFound, dd_RegionEmpty
} dd_CompStatusType;

typedef enum {
  dd_DimensionTooLarge, dd_ImproperInputFormat,
  dd_NegativeMatrixSize, dd_EmptyVrepresentation,
  dd_IFileNotFound, dd_OFileNotOpen, dd_NoLPObjective, dd_NoRealNumberSupport, dd_NoError
} dd_ErrorType;

typedef enum {
  dd_LPnone=0, dd_LPmax, dd_LPmin
} dd_LPObjectiveType;
```

```
typedef enum {
  dd_LPSundecided, dd_Optimal, dd_Inconsistent, dd_DualInconsistent,
  dd_StrucInconsistent, dd_StrucDualInconsistent,
  dd_Unbounded, dd_DualUnbounded
} dd_LPStatusType;

typedef struct matrixdata *dd_MatrixPtr;
typedef struct matrixdata {
  dd_rowrange rowsize;
  dd_rowset linset;
    /*  a subset of rows of linearity (ie, generators of
        linearity space for V-representation, and equations
        for H-representation. */
  dd_colrange colsize;
  dd_RepresentationType representation;
  dd_NumberType numbtype;
  dd_Amatrix matrix;
  dd_LPObjectiveType objective;
  dd_Arow rowvec;
}  dd_MatrixType;

typedef struct setfamily *dd_SetFamilyPtr;
typedef struct setfamily {
  dd_bigrange famsize;
  dd_bigrange setsize;
  dd_SetVector set;
} dd_SetFamilyType;

typedef struct lpsolution *dd_LPSolutionPtr;
typedef struct lpsolution {
  dd_DataFileType filename;
  dd_LPObjectiveType objective;
  dd_LPSolverType solver;
  dd_rowrange m;
  dd_colrange d;
  dd_NumberType numbtype;

  dd_LPStatusType LPS;  /* the current solution status */
  mytype optvalue;  /* optimal value */
  dd_Arow sol;   /* primal solution */
  dd_Arow dsol;  /* dual solution */
  dd_colindex nbindex;  /* current basis represented by nonbasic indices */
  dd_rowrange re;  /* row index as a certificate in the case of inconsistency */
  dd_colrange se;  /* col index as a certificate in the case of dual inconsistency */
  long pivots[4];
   /* pivots[0]=setup (to find a basis), pivots[1]=PhaseI or Criss-Cross,
      pivots[2]=Phase II, pivots[3]=Anticycling */
  long total_pivots;
```

```
} dd_LPSolutionType;
```

4 Library Functions

Here we list some of the most important library functions/procedures. We use the following convention: `poly` is of type `dd_PolyhedraPtr`, `matrix`, `matrix1` and `matrix2` are of type `dd_MatrixPtr`, `err` is of type `dd_ErrorType*`, `ifile` and `ofile` are of type `char*`, `A` is of type `dd_Amatrix`, `point` and `vector` are of type `dd_Arow`, `d` is of type `dd_colrange`, `m` and `i` are of type `dd_rowrange`, `x` is of type `mytype`, `a` is of type `signed long integer`, `b` is of type `double`, `set` is of type `set_type`. Also, `setfam` is of type `dd_SetFamilyPtr`, `lp` is of type `dd_LPPtr`, `solver` is of type `dd_LPSolverType`.

4.1 Library Initialization

`void dd_set_global_constants(void) :`

This is to set the global constants such as `dd_zero`, `dd_purezero` and `dd_one` for sign recognition and basic arithmetic operations. Every program to use cddlib must call this function before doing any computation. Just call this once.

4.2 Core Functions

There are two types of core functions in cddlib. The first type runs the double description (DD) algorithm and does a representation conversion of a specified polyhedron. The standard header for this type is `dd_DD*`. The second type solves an linear program and the standard naming is `dd_LP*`. Both computations are nontrivial and the users (especially for the DD algorithm) must know that there is a serous limit in the sizes of problems that can be practically solved. Please check *.ext and *.ine files that come with cddlib to get ideas of tractable problems.

`dd_PolyhedraPtr dd_DDMatrix2Poly(matrix, err) :`

Store the representation given by `matrix` in a polyhedra data, and generate the second representation of `*poly`. It returns a pointer to the data. `*err` returns `dd_NoError` if the computation terminates normally. Otherwise, it returns a value according to the error occured.

`boolean dd_DDInputAppend(&poly, matrix, &err) :`

Modify the input representation in `*poly` by appending the matrix of `*matrix`, and compute the second representation. The number of columns in `*matrix` must be equal to the input representation.

`boolean dd_LPSolve(lp, solver, &err) :`

Solve `lp` by the algorithm `solver` and save the solututions in `*lp`. Unlike the earlier versions (dplex, cdd+), it can deal with equations and totally zero right hand sides.

`dd_boolean dd_Redundant(matrix, i, point, &err) :`

Check whether ith data in `matrix` is redundant for the representation. If it is nonredundant, it returns a certificate. For H-representation, it is a `point` in R^d which satisfies all inequalities except for the ith inequality. If i is a linearity, it does nothing and always returns FALSE.

`dd_rowset dd_RedundantRows(matrix, &err) :`
Returns a maximal set of row indices such that the associated rows can be eliminated without changing the polyhedron. The function works for both V- and H-representations.

`dd_boolean dd_ImplicitLinearity(matrix, i, &err) :`
Check whether ith inequality in the input is forced to be linearity (equality for H-representation). If i is linearity itself, it does nothing and always returns FALSE.

`dd_rowset dd_ImplicitLinearityRows(matrix, &err) :`
Returns the set of indices of rows that are implicitly linearity. It simply calls `dd_ImplicitLinearity` for each inequality and collects the row indices for which the answer is TRUE.

`dd_rowrange dd_RayShooting(matrix, point, vector) :`
Finds the index of a halfspace first left by the ray starting from `point` toward the direction `vector`. It resolves the tie by a lexicographic perturbation. Those inequalities violated by `point` will be simply ignored.

4.3 Data Manipulations

4.3.1 Number Assignments

`void dd_set_si(x, a) :`
This is to set a `mytype` variable `x` to the value of signed long integer `a`. One cannot use such expressions as `x=(mytype)a`. This is a price to pay for the generality: cddlib uses an abstract number type (`mytype`) so that it can compute with various number types such as C double and GMP rational. User can easily add a new number type by redefining arithmetic operations in cddmp.h and cddmp.c.

`void dd_set_d(x, b) :`
This is to set a `mytype` variable `x` to the value of double `b`. This is available only when the library is compiled without `-DGMPRATIONAL` compiler option.

4.3.2 Polyhedra Data Manipulation

`dd_MatrixPtr dd_PolyFile2Matrix (f, err) :`
Read a Polyhedra data from `f` and store it in `matrixdata` and return a pointer to the data.

`dd_MatrixPtr dd_CopyInequalities(poly) :`
Copy the inequality representation pointed by poly to `matrixdata` and return `dd_MatrixPtr`.

`dd_MatrixPtr dd_CopyGenerators(poly) :`
Copy the generator representation pointed by poly to `matrixdata` and return `dd_MatrixPtr`.

`dd_SetFamilyPtr dd_CopyIncidence(poly) :`
Copy the incidence representation of the computed representation pointed by poly to `setfamily` and return `dd_SetFamilyPtr`. The computed representation is `Inequality` if the input is `Generator`, and the vice visa.

`dd_SetFamilyPtr dd_CopyAdjacency(poly) :`
Copy the adjacency representation of the computed representation pointed by poly to `setfamily` and return `dd_SetFamilyPtr`. The computed representation is `Inequality` if the input is `Generator`, and the vice visa.

`dd_SetFamilyPtr dd_CopyInputIncidence(poly) :`
: Copy the incidence representation of the input representation pointed by poly to `setfamily` and return `d_SetFamilyPtr`.

`dd_SetFamilyPtr dd_CopyInputAdjacency(poly) :`
: Copy the adjacency representation of the input representation pointed by poly to `setfamily` and return `d_SetFamilyPtr`.

`void dd_FreePolyhedra(poly) :`
: Free memory allocated to `poly`.

4.3.3 LP Data Manipulation

`dd_LPPtr dd_MakeLPforInteriorFinding(lp) :`
: Set up an LP to find an interior point of the feasible region of `lp` and return a pointer to the LP. The new LP has one new variable x_{d+1} and one more constraint: $\max x_{d+1}$ subject to $b - Ax - x_{d+1} \geq 0$ and $x_{d+1} \leq K$, where K is a positive constant.

`dd_LPPtr dd_Matrix2LP(matrix, err) :`
: Load `matrix` to `lpdata` and return a pointer to the data.

`dd_LPSolutionPtr dd_CopyLPSolution(lp) :`
: Load the solutions of `lp` to `lpsolution` and return a pointer to the data. This replaces the old name `dd_LPSolutionLoad(lp)`.

`void dd_FreeLPData(lp) :`
: Free memory allocated to `lp`.

4.3.4 Matrix Manipulation

`dd_MatrixPtr dd_CopyMatrix(matrix) :`
: Make a copy of matrixdata pointed by `matrix` and return a pointer to the copy.

`dd_MatrixPtr dd_AppendMatrix(matrix1, matrix2) :`
: Make a matrixdata by copying `*matrix1` and appending the matrix in `*matrix2` and return a pointer to the data. The colsize must be equal in the two input matrices. It returns a `NULL` pointer if the input matrices are not appropriate. Its `rowsize` is set to the sum of the rowsizes of `matrix1` and `matrix2`. The new matrixdata inherits everything else (i.e. numbertype, representation, etc) from the first matrix.

`int dd_MatrixAppendTo(& matrix1, matrix2) :`
: Same as `dd_AppendMatrix` except that the first matrix is modified to take the result.

`int dd_MatrixRowRemove(& matrix, i) :`
: Remove the ith row of `matrix`.

`dd_MatrixPtr dd_MatrixSubmatrix(matrix, set) :`
: Generate the submatrix of `matrix` by removing the rows indexed by `set` and return a matrixdata pointer.

4.4 Input/Output Functions

`boolean dd_DDFile2File(ifile, ofile, err) :`
Compute the representation conversion for a polyhedron given by a Polyhedra file ifile, and write the other representation in a Polyhedra file ofile. `*err` returns `dd_NoError` if the computation terminates normally. Otherwise, it returns a value according to the error occured.

`void dd_WriteMatrix(f, matrix) :`
Write `matrix` to stream `f`.

`void dd_WriteNumber(f, x) :`
Write `x` to stream `f`. If `x` is of GMP mpq_t rational p/q, the output is p/q. If it is of C double, it is formated as a double float with a decimal point.

`void dd_WritePolyFile(f, poly) :`
Write tt poly to stream `f` in Polyhedra format.

`void dd_WriteErrorMessages(f, err) :`
Write error messages given by `err` to stream `f`.

`void dd_WriteSetFamily(f, setfam) :`
Write the set family pointed by `setfam` to stream `f`. For each set, it outputs its index, its cardinality, a colon ":" and a ordered list of its elements.

`void dd_WriteSetFamilyCompressed(f, setfam) :`
Write the set family pointed by `setfam` to stream `f`. For each set, it outputs its index, its cardinality or the negative of the cardinality, a colon ":" and the elements in the set or its complements whichever is smaller. Whenever it outputs the complements, the cardinality is negated so that there is no ambiguity. This will be considered standard for outputing incidence (*.icd, *ecd) and adjacency (*.iad, *.ead) data in cddlib. But there is some minor incompatibility with cdd/cdd+ standalone codes.

`void dd_WriteProgramDescription(f) :`
Write the cddlib version information to stream `f`.

`void dd_WriteDDTimes(f, poly) :`
Write the representation conversion time information on `poly` to stream `f`.

4.5 Obsolete Functions

`boolean dd_DoubleDescription(poly, err) :` (removed in Version 0.90c)
The new function `dd_DDMatrix2Poly(matrix, err)` (see Section 4.2) replaces (and actually combines) both this and `dd_Matrix2Poly(matrix, err)`.

`dd_PolyhedraPtr dd_Matrix2Poly(matrix, err) :` (removed in Version 0.90c)
See above for the reason for removal.

`dd_LPSolutionPtr dd_LPSolutionLoad(lp) :` (renamed in Version 0.90c)
This function is now called `dd_CopyLPSolution(lp)`.

5 How to Use

For the moment, the best documentation of usage are the examples, testcdd*.c , testlp*.c, redcheck.c and simplecdd.c.

6 Numerical Accuracy

A little caution is in order. Many people have observed numerical problems of cddlib when the floating version of cddlib is used. As we all know, floating-point computation might not give a correct answer, especially when an input data is very sensitive to a small perturbation. When some strange behavior is observed, it is always wise to create a rationalization of the input (for example, one can replace 0.3333333 with 1/3) and to compute it with cddlib compiled with gmp rational to see what a correct behavior should be. Whenever the time is not important, it is safer to use gmp rational arithmetic.

If you need speedy computation with floating-point arithmetic, you might want to "play with" the constant `dd_almostzero` defined in cdd.h:

```
#define dd_almostzero  1.0E-7
```

This number is used to recognize whether a number is zero: a number whose absolute value is smaller than `dd_almostzero` is considered zero, and nonzero otherwise. You can change this to modify the behavior of cddlib. One might consider the default setting is rather large for double precision arithmetic. This is because cddlib is made to deal with highly degenerate data and it works better to treat a relatively large "epsilon" as zero.

Another thing one can do is scaling. If the values in one column of an input is of smaller magnitude than those in another column, scale one so that they become comparable.

7 FTP site

An anonymous ftp site for the programs is set at:

```
ftpsite:  ftp.ifor.math.ethz.ch
directory: pub/fukuda/cdd
filenames: cddlib-***.tar.gz
```

Since the file is compressed binary file, it is necessary to use binary mode for file transfer.

8 Other Useful Codes

There are several other useful codes available for vertex enumeration and/or convex hull computation such as lrs, qhull, porta and irisa-polylib. The pointers to these codes are available at

1. lrs by D. Avis [2] (C implementation of the reverse search algorithm [4]).

2. qhull by C.B. Barber [5] (C implementation of the beneath-beyond method, see [8, 15], which is the dual of the dd method).

3. porta by T. Christof and A. Löbel [7] (C implementation of the Fourier-Motzkin elimination).

4. IRISA polyhedral library by D.K. Wilde [16] (C implementation of a variation of the dd algorithm).
5. pd by A. Marzetta [13] (C implementation of the primal-dual algorithm [6]).
6. Geometry Center Software List by N. Amenta [1].
7. Computational Geometry Pages by J. Erickson [9].
8. Linear Programming FAQ by R. Fourer and J. Gregory [10].
9. ZIB Berlin polyhedral software list: ftp://elib.zib-berlin.de/pub/mathprog/polyth/index.html.
10. Polyhedral Computation FAQ [11].

Acknowledgements.

I am grateful to Th. M. Liebling who provided me with an ideal opportunity to visit EPFL for the academic year 1993-1994. Without his support, the present form of this program would not have existed. Later, H.-J. Lüthi (ETHZ) joined to support the the development of cdd codes (cdd, cdd+, cddlib). There are many people who helped me to improve cdd, in particular, I am indebted to David Avis, Alexander Bockmayr, David Bremner, Henry Crapo, Istvan Csabai, Francois Margot, Marc Pfetsch, Alain Prodon, Jörg Rambau, Shawn Rusaw, Matthew Saltzman, Masanori Sato.

References

[1] N. Amenta. Directory of computational geometry. http://www.geom.umn.edu/software/cglist/.

[2] D. Avis. *User's Guide for lrs - Version 3.2*, 1997. available from lrs homepage ftp://mutt.cs.mcgill.ca/pub/C/lrs.html.

[3] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms. *Computational Geometry: Theory and Applications*, 7:265–302, 1997.

[4] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.*, 8:295–313, 1992.

[5] C.B. Barber, D.P. Dobkin, and H. Huhdanpaa. *qhull, Version 2.1.* The Geometry Center, Minnesota, U.S.A., 1995. program and report available from ftp://geom.umn.edu/pub/software/qhull.tar.Z.

[6] D. Bremner, K. Fukuda, and A. Marzetta. Primal-dual methods for vertex and facet enumeration. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 49–56, 1997.

[7] T. Christof and A. Löbel. PORTA: Polyhedron representation transformation algorithm (ver. 1.3.1), 1997. http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/PORTA/readme.html.

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

[9] J. Erickson. Computational geometry pages, list of software libraries and codes. http://compgeom.cs.uiuc.edu/ jeffe/compgeom/.

[10] R. Fourer and J.W. Gregory. Linear programming frequently asked questions (LP-FAQ), 1997. http://www.mcs.anl.gov/home/otc/Guide/faq/linear-programming-faq.html.

[11] K. Fukuda. Polyhedral computation FAQ, 1998. Both html and ps versions available from http://www.ifor.math.ethz.ch/staff/fukuda/fukuda.html.

[12] K. Fukuda and A. Prodon. Double description method revisited. In M. Deza, R. Euler, and I. Manoussakis, editors, *Combinatorics and Computer Science*, volume 1120 of *Lecture Notes in Computer Science*, pages 91–111. Springer-Verlag, 1996. ps file available from ftp://ftp.ifor.math.ethz.ch/pub/fukuda/reports/ddrev960315.ps.gz.

[13] A. Marzetta. *pd – C-implementation of the primal-dual algoirithm*, 1997. code available from http://wwwjn.inf.ethz.ch/ambros/pd.html.

[14] T.S. Motzkin, H. Raiffa, GL. Thompson, and R.M. Thrall. The double description method. In H.W. Kuhn and A.W.Tucker, editors, *Contributions to theory of games, Vol. 2.* Princeton University Press, Princeton, RI, 1953.

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

[16] D.K. Wilde. A library for doing polyhedral operations. Master's thesis, Oregon State University, Corvallis, Oregon, Dec 1993. Also published in IRISA technical report PI 785, Rennes, France; Dec, 1993.