cdd+ Reference Manual

Komei Fukuda
Institut fur Operations Research
ETH Zentrum, CH-8092 Zurich, Switzerland
fukuda@ifor.math.ethz.ch
and
University of Tsukuba, Tokyo, Japan
fukuda@gssm.otsuka.tsukuba.ac.jp

(cdd+ Version 0.73, September 6, 1995)
(Manual Version September 6, 1995)

cdd+ Reference Manual

Komei Fukuda
fukuda@ifor.math.ethz.ch or fukuda@gssm.otsuka.tsukuba.ac.jp
Institut fur Operations Research, ETH Zentrum, CH-8092 Zurich, Switzerland,
and University of Tsukuba, Tokyo, Japan

(cdd+ Version 0.73, September 6, 1995)

1 Introduction

The program cdd+ (cdd, respectively) is a C++ (ANSI C) implementation of the Double Description Method [MRTT53] for generating all vertices (i.e. extreme points) and extreme rays of a general convex polyhedron given by a system of linear inequalities:

$$P = \{x \in R^d : Ax \leq b\}$$

where A is an $m \times d$ real matrix and b is a real m dimensional vector.

The program cdd+ is a C++ program converted from the ANSI C program cdd. One major advantage of this C++-version over the C version is that it can be compiled for both rational (exact) arithmetic and floating point arithmetic. Note that cdd runs on floating arithmetic only. Since cdd+ uses GNU g++ library, in particular Rational library, one needs a recent (2.6.0 or higher) gcc compiler and g++-lib. One should be also warned that the computation can be considerably (10 - 100 times) slower if the rational arithmetic is used.

One useful feature of cdd+ (and cdd) is its capability of handling the dual (reverse) problem without any transformation of data. The dual problem is known to be the *(convex) hull problem* which 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: $P = conv(v_1, \ldots, v_n) + nonneg(r_1, \ldots, r_s)$. As we see in this manual, the computation can be done in straightforward manner. There is one assumption for the input for hull computation: the polyhedron must be full-dimensional. See the "hull" option in Section 3.

The program cdd+ (and cdd) reads input and writes output in *Polyhedra format* which was defined by David Avis and the author. The program called rs developed by David Avis is a C-implementation of the reverse search algorithm [AF92] 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.

Among the hardest problems that could be solved (in floating-point arithmetic) by cdd+ is a 21-dimensional hull problem given by 64 vertices. This polytope, known as the *complete cut polytope on 7 points*, has exactly 116,764 facets and some of facets contain many vertices. It took 205 hours (eight and half days!) for cdd to compute the facets exactly on a SUN SparkServer 1000. The input file (ccp7.ine) of this polytope is included in the distribution. A considerably easier problem is ccc7.ine which is a variation of the problem (see e.g. [G90]).

The size of an input file hardly indicates the degree of hardness of its vertex/ray enumeration. While this program can handle a highly degenerate problem (prodmT5.ine) with 711 inequalities in 19 dimension quite easily with the computation time 1-2 minutes on a fast workstation, a

8-dimensional problem (mit729-9.ine) with 729 inequalities can be extremely hard. It takes two days to compute all (only 4862) vertices by a fast SUN SparkServer 1000. The latter problem arises from the ground state analysis of a ternary alloy model, see [CGAF93]. Both input files are included in the distribution.

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 rs or qhull (developed by the Geometry Center), might be more efficient. See Section 9 for pointers to these codes.

This program can be distributed freely under the GNU GENERAL PUBLIC LICENSE. Please read the file COPYING carefully before using.

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 to me what purposes cdd has been used for. I will be happy to include an application description in future distribution if I receive enough replies. The most powerful support for free software development is user's appreciation and collaboration.

2 Polyhedra Input and Output Format

Polyhedra input format is quite simple:

```
various comments
begin
m   d+1   numbertype
 b    -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). There is one restriction in the current polyhedra format: the last d rows must determine a vertex of P. The program cdd+ does not care whether this condition is satisfied, as long as the polyhedron contains at least one vertex. But one can specify that the last $(d+1)$ rows be chosen as the initial set of $(d+1)$ rows for the double description algorithm. See **initbasis_at_bottom** option in Section 3.

Polyhedra output format is quite similar to input format:

various comments produced by a program
begin
$n+s \quad d+1$ **numbertype**
$1 \quad v_1$
$\vdots \quad \vdots$
$1 \quad v_n$
$0 \quad r_1$
$\vdots \quad \vdots$
$0 \quad r_s$
end

where $v_1, \ldots, v_n$ are the vertices and $r_1, \ldots, r_s$ are the extreme rays of the polyhedron P. Here we do not require that the vertex list and the ray list are output separately; they can appear mixed in arbitrary order.

For example, let P be the following unbounded 3-dimensional polyhedron given by

$$P = \{x \in R^3 : 1 \leq x_1 \leq 2,\ 1 \leq x_2 \leq 2,\ 1 \leq x_3\},$$

which is a 3-cube without one "lid". For finding all vertices and extreme rays, the input file for cdd+ is

```
file name: ucube.ine
3 cube without one "lid"
begin
    5      4    integer
  2   -1   0   0
  2    0  -1   0
 -1    1   0   0
 -1    0   1   0
 -1    0   0   1
end
incidence
adjacency
input_adjacency
```

The meaning of options "incidence", "adjacency" and "input_adjacency" will be explained in Section 3. After you run cddf+ (the floating-arithmetic version of cdd+) with this input file, you will get an output file, say ucube.ext, which looks like:

```
* cdd+: Double Description Method in C++:Version 0.73 (September 6, 1995)
* Copyright (C) 1995, Komei Fukuda, fukuda@ifor.math.ethz.ch
* Compiled for Floating-Point Arithmetic
*Input File:ucube.ine(5x4)
*HyperplaneOrder: LexMin
*Degeneracy preknowledge for computation: None (possible degeneracy)
*Vertex/Ray enumeration is chosen.
*Computation starts     at Fri Aug 11 14:46:45 1995
*            terminates at Fri Aug 11 14:46:45 1995
```

```
*Total processor time = 0 seconds
*                     = 0h 0m 0s
*FINAL RESULT:
*Number of Vertices =4, Rays =1
begin
5  4  real
 1 2 1 1
 1 1 1 1
 1 1 2 1
 1 2 2 1
 0 0 0 1
end
hull
```

The output shows that the polyhedron has four vertices $(2,1,1)$, $(1,1,1)$, $(1,2,1)$, $(2,2,1)$ and only one extreme ray $(0,0,1)$. The comments contain information on the name of input file, and the options chosen to run the program which will be explained in the next section.

Finally the "hull" option is set for the output file so that if you run cdd+ with this output file, cdd+ will perform the convex hull operation to recover essentially the original input inequality system. Note that this back-and-forth transformation of a polyhedron works only when the polyhedron is full dimensional and contans at least one vertex.

3 Options

The following options are available for cdd. These options are set if they appear in input file after the "end" command. Independent options can be set simultaneously, but each option must be written separately in one line, and two options should not be written in one line. When two or more non-independent options are specified, the last one overrides the others.

hull option

When this option is chosen, the program cdd will do the reverse operation. That is, the input is assumed to be a set of points and directions (rays). When this option is set, it is required that each data line must start with either "1" or "0", meaning points and rays, respectively. More specifically, if the input file is of form

comments		
begin		
$n+s$	$d+1$	**numbertype**
1	v_1	
$\vdots$	$\vdots$	
1	v_n	
0	r_1	
$\vdots$	$\vdots$	
0	r_s	
end		
hull		

Then the input is interpreted as the polyhedron in R^d:
$P = conv(v_1, \dots, v_n) + nonneg(r_1, \dots, r_s)$ and the output will be a minimal system of linear inequalities to represent P.

verify_input option
When this option is chosen, the program will output the input problem as cdd+ interpreted. The default output file is "*.solved". This option helps user to verify what problem is actually solved. The default for this option is off. See the sample files verifyinput1.ine and verifyinput2.ine

dynout_off option
When this option is chosen, the program will not output vertices and rays to the CRT in real time. The default is dynout_on.

stdout_off option
When this option is chosen, the program will not output any progress report of computation (iteration number. etc). The default is stdout_on.

logfile_on option
When this option is chosen, the program will output to a specified file (*.ddl) some information on the computation history. This can be useful when the user does not know which hyperplane order (mincutoff, maxcutoff, mixcutoff, lexmin, lexmax, minindex, random) is efficient for computation.

incidence, #incidence options
When the **incidence** option is selected, the incidence relations between the vertices/rays and the inequalities will be output in a separate file (*.icd). Here, a vertex is said to be *incident with* an inequality if the inequality is satisfied by equality. An extreme ray r is said to be *incident with* an inequality $a^T\ x \leq b$ if $a^T\ r = 0$. For example, since the incidence option was set for the example input file ucube.ine in the previous section, the program outputs the following ucube.icd file:

```
*Incidences of output(=vertices/rays) and input (=hyperplanes)
*   for each output, #incidence and the set of hyperplanes containing it
*   or its complement with its cardinality with minus sign
*cdd input file : ucube.ine  5  4
*cdd output file: ucube.ext
*After <begin>, three numbers are output_size, m and m1,
*where m1 is m+1 (for vertex/ray enumeration) or m (for convex hull).
begin
  5  5  6
 3 :   1 4 5
 3 :   3 4 5
 3 :   2 3 5
 3 :   1 2 5
 -1 :   5
end
```

After "begin", there are three numbers 5   5   6. The first number 5 is a number of output (vertices and rays). The next number 5 is m, the number of inequalities in the input file. The last number 6 is usually $m + 1$, and m if the input linear inequality system is homogeneous (i.e., has zero RHS) or the hull option is chosen. The number $m + 1$ corresponds to the infinity constraint which is added for vertex/ray enumeration when the input system is not homogeneous.

The incidence data starts right after these three numbers. At each line, the cardinality of incident inequalities and the list of their indices are given. There is an exception that, when there are more incident inequalities than non-incident ones, then the program outputs the list of non-incident inequalities with its size with negative sign. This is to save space of output.

For example, the first output line 3 : 1 4 5 corresponds to the first vertex of ucube.ext file in previous section, that is, the vertex $(2, 1, 1)$. The first number 3 is simply the number of incident inequalities and the rest is the indices of those inequalities, and so the 1st, 4th and 5th inequalities are satisfied by equality at this vertex. The last output -1 : 5 corresponds to the ray $(0, 0, 1)$. Since all inequalities except the last (5th) inequality are incident with this ray, the output is the complementary list with its cardinality (=1) with negative sign. Note that the full list would be 5 : 1 2 3 4 6, where 6 is the infinity plane. One can ignore the infinity plane for some purposes, but for analyzing the combinatorial structure of polyhedra, it is very important information.

The **#incidence** option can be used when you do not wish to output the incidence file but to output only the cardinality of incidence for each output, at the end of each output line.

The incidence file (adjacency file, input_adjacency as well) can be created independently after *.ext file is created, see "postanalysis" option.

nondegenerate option

When this option is set, the program assumes that the input system is not degenerate, i.e., there is no point in the space R^d satisfying more than d inequalities of input with equality. It will run faster with this option, but of course, if this option is set for degenerate inputs, it is quite possible that the output is incorrect. The default is this option being off.

adjacency option

This option can be used when you want to output the adjacency of output. When the output is the list of vertices and rays, the program will output the adjacency list. For the example input "ucube.ine", the following extra file, say "ucube.adj", will be created:

```
*Adjacency List of output (=vertices/rays)
*cdd input file : ucube.ine (5 x 4)
*cdd output file: ucube.ext
begin
  5
 1 3 : 2 4 5
 2 3 : 1 3 5
 3 3 : 2 4 5
 4 3 : 1 3 5
 5 4 : 1 2 3 4
end
```

The first number 5 is simply the number of outputs of cdd, the number of vertices and rays in this case. The second line 1 3 : 2 4 5 says that the first output of ucube.ext file has degree (valency) 3, and its three neighbours are 2nd, 4th and 5th output.

When the computation is to obtain the hull (inequality system), the adjacency is of course that of inequalities (i.e. facets).

The adjacency file (incidence file, input_adjacency file) can be created independently after *.ext file is created, see "postanalysis" option.

input_adjacency option

This option is for outputing the adjacency of input inequalities. Here, two inequalities are defined to be *adjacent* if they are nonredundant and there is no third input inequality which is satisfied with equality at all points of the polyhedron that satisfy the two inequalities with equality. In more intuitive language, two inequalities are adjacent if each determine a facet of the polyhedron and the intersection of the two facets is not contained in any other facet.

The default file name for this output is *.iad. This file lists the redundancy information of input also. For the example "ucube.ine" above, the following "ucube.iad" will be generated:

```
*Adjacency List of input (=inequalities/facets)
*cdd input file : ucube.ine (5 x 4)
*cdd output file: ucube.ext
*row 6 is redundant;dominated by: 1 2 3 4
begin
  6
 1 3 : 2 4 5
 2 3 : 1 3 5
 3 3 : 2 4 5
 4 3 : 1 3 5
 5 4 : 1 2 3 4
 6 0 :
end
```

Observe that the artificially added 6th inequality (infinity) is redundant because the first four facets intersects at a single infinity point (corresponding to a unique extreme ray) and hence the polyhedron has no infinity facet, although the polyhedron is not bounded.

The input_adjacency file can be created independently after *.ext file is created, see "post-analysis" option.

postanalysis option

It is often more desireble to compute the adjacency, input_adjacency and incidence relations independently from the main (and often heavy) computation of enumerating all vertices and extreme rays. The "postanalysis" option can be used together with "adjacency" and/or "incidence" options for this purpose to create *.adj and/or *.icd files from both *.ine and *.ext files. If *.ine file contains this option, cdd+ will open the corresponding *.ext file and output requested *.adj, *.iad and/or *.icd files. An error occurs when *.ext file does not exist in the current directory.

lexmin, lexmax, minindex,mincutoff, maxcutoff, mixcutoff, random options

The double description is an incremental algorithm which computes the vertices/rays of a polyhedron given by some k of original inequalities from the precomputed vertices/rays of a polyhedron given by $k-1$ inequalities. It is observed that the efficiency of the algorithm depends strongly on how one selects the ordering of inequalities, although a little can be said theoretically. These options are to select the ordering of inequalities to be added at each iteration, and it is recommended to do small experiment to select good ordering for a specific type of problems. Unfortunately, a good ordering depends on the problem and there does not seem to be THE BEST ordering for every computation. From our experiences, lexmin, lexmax, mincutoff, maxcuoff work quite well in general.

The default is lexmin ordering which simply order inequalities with respect to lexicographic ordering of rows of $(b, -A)$. The lexmax is reverse of lexmin. The mincutoff (maxcutoff) option selects an inequality which cuts off the minimum (maximum) number of vertices/rays of the $(k-1)$st polyhedron. The mixcutoff option is the mixture of mincutoff and maxcutoff which selects an inequality which cuts off the $(k-1)$st polytope as unbalanced as possible. The maxcutoff option might be efficient if the input contains many redundant inequalities (many interior points for hull computation). The minindex option selects the hyperplanes from the top of the input.

The random option selects the inequalities in a random order. This option must be followed by a random seed which is positive integer (less than 65536). For example, **random 123** specifies the random option with the random seed 123.

initbasis_at_bottom option

When this option is set, the program tries to select the initial set of rows for the double description method from the bottom of the input. This means that if the last (d+1) rows are independent, they will be chosen to initiate the algorithm.

This option is *not* default. The default follows the same ordering as the ordering of inequalities chosen. This means that if **lexmin** is the ordering of inequalities, then the initial independent rows will be chosen sequentially with lexico-min ordering. There are exceptions when this rule is not applicable, i.e. when one of mincutoff or maxcutoff options is chosen. In such cases, **lexmin** ordering will be chosen.

maximize, minimize options

When maximize option is set with an objective vector $c_0\ c_1\ c_2 \ldots c_d$, the program simply solves the linear program: $\max c_0 + c_1x_1 + c_2x_2 + \cdots + c_dx_d$ over the input polyhedron P. The grammer is simply

```
various comments
begin
m   d+1   numbertype
 b    -A
end
maximize
c0  c1  c2   ...   cd
```

The minimize option works exactly same way for minimization of a linear objective function. See the sample input file "lptest.ine". The program cdd will output both primal and dual optimal solutions if the LP is solvable. If the LP is infeasible (dual infeasible), then it will output an evidence.

For the moment, the LP solver is primitive: one can use either the dual simplex method (option "dual-simplex", default) or the criss-cross method by Terlaky-Wang. The latter method can be specified by option "criss-cross" and is very sensible to the ordering of inequalities. The ordering options such as maxindex, lexmin and random will affect the behavior of this solver. Try to use a different ordering, if the computation takes too much time.

Also, in order to see the intermediate LP sign tableau one can use "show_tableau" option. Also use "manual_pivot" option to select pivots manually. Of course, these options are intended for very small problems.

find_interior option

When this option is set, the program solves the linear program: $\max x_{d+1}$ subject to $Ax + ex_{d+1} \leq b$, where e is the column vector of all 1's. If the optimum value is zero, the polyhedron has no interior point. If the optimum value is negative then the polyhedron is empty. If the LP is dual inconsistent, then the polyhedron admits unbounded inscribed balls. To find any interior point in this last case, one must add some inequality(ies) to bound the polyhedron.

facet_listing option

When this option is set, the program checks for each i-th row of the input whether the associated inequality $A_i x \leq b_i$ determines a facet of the polyhedron.

tope_listing option

When this option is set, the program generates all full-dimensional regions (which are sometimes called topes) of the arrangement of hyperplanes $\{h_i : i = 1, 2, \ldots, m\}$, where $h_i = \{x : A_i x \leq b_i\}$. Each tope will be represented by its location vector, i.e. a sign vector in $\{+, -\}^m$ whose i-component indicates the (positive or negative) side of the hyperplane h_i the tope is located. This procedure assumes that the input polyhedron is full-dimensional and thus the vector of all +'s determines a tope.

partial_enumeration, equality, strict_inequality options

With partial_enumeration option (or equivalently equality option), one can enumerate only those vertices and rays that are lying on the set of hyperplanes associated with specified inequality numbers. If you want to compute all vertices/rays lying on hyperplanes associated with k inequalities $i_1, i_2, \ldots, i_k$ $(1 \leq i_j \leq m)$, then the option should be specified as

---

various comments
begin
$m \quad d+1$ **numbertype**
$b \quad -A$
end
partial_enumeration
$k \quad i_1 \quad i_2 \quad \cdots \quad i_k$

The **strict_inequality** option follows the same grammer as partial_enumeration or equality. With this, cdd outputs only those vertices and rays not satisfying any of the specified inequalities with equality. See the sample files, partialtest1.ine and partialtest2.ine.

These options make no effect on LP maximization or minimization.

preprojection option

This option is for a preprocessing of orthogonal projection of the polyhedon to the subspace of R^d spanned by a subset of variables. That is, if the inequality inequality system is of two-block form $A_1x_1 + A_2x_2 \leq b$, and the variable indices for x_1, say $1, 4, 6, 7$, are listed in the input file as

```
begin
m   d+1   numbertype
b   -A
end
preprojection
4  1  4  6  7
```

Then, cdd+ will output the inequality system, $A_1x_1 \leq b$, together with the list R of extreme rays of the homogeneous cone $\{z : z \geq 0 \text{ and } z^TA_2 = 0\}$. Consequently, the inequality system $\{\ \ r^TA_1x_1 \leq r^Tb :\ \ \ r \in R\}$ represents the projection of the original polyhedron onto x_1-space with possible redundancy. The default file names for the inequality system output and the extemal ray output are *sub.ine and *.ext, respectively if the input file is named *.ine.

There is a supplementary C program, called domcheck, written by F. Margot, EPFL, which generates quickly a minimal (i.e. nonredundant) system from these two outputs. This program can be obtained from the standard ftp site for cdd.

4 How to Use

The program hardly has any user interface. Once you have compiled executable files, *cddf+* and *cddr+* (see Section 5), and once you create an input file, say, *test.ine*, you have basically two ways to run the program. The simplest way is just to run the program with

```
  % cddf+ test.ine
```

or, if you want to compute with rational (exact arithmetic)

```
  % cddr+ test.ine
```

Then the program will open necessary output files with default file names, and output the requested results. The default names are *test.ext*, *test.solved*, *test.icd*, *test.adj*, *test.iad*, *testsub.ine* for the extreme points/ray file, the input verification file, the incidence file, the adjacency file, the input_adjacency and the preprojection variable sub-inequality system, respectively. If you want to specify the output file names different from default, simply run the program by

```
  % cdd
```

and input desired file names at each of file name requests. Even after you run cdd this way, one can change to the automatic mode by inputing the input file name with additional semicolon, e.g. "test.ine;".

5 Source Files and Compilation

(1) [Files and Compilation] The source files for distribution are

cdd+.readme	The readme file
cdd.C	C++ main source file
cddarith.C	C++ main arithmetic code
cddpivot.C	C++ pivot operation arithmetic code
cddio.C	C++ IO code
cddrevs.C	C++ reverse search code
cdd.h	The header file for cdd.C
cdddef.h	cdd+ definition file (whose two lines are to be edited by user)
cddtype.h	cdd+ arithmetic type definition file
cddrevs.h	The header file for cddrevs.C
setoper.C	C++ library for set operation
setoper.h	The header file for setoper.C
cddman.tex	Latex source file of cdd+ Reference Manual
cddHISTORY	brief description of changes made at each updates
ine	A subdirectory containing sample input files
ext	A subdirectory containing sample output files
COPYING	GNU GENERAL PUBLIC LICENSE

For compilation of cdd+, one needs a recent (2.6.0 or higher) gcc compiler and g++-library. Once gcc and g++-library are installed, please edit Makefile according to the setup of a GNU gcc compiler and g++-library, and type

```
% make all
```

which creates two executables, cddr+ and cddf+. The executable cddr+ computes with rational (exact arithmetic) and cddf+ computes with floating-point arithmetic. If you want to create only one of them, use "make cddf+" or "make cddr+". Once these executables are created one might want to remove all object files *.o by

```
% rm *.o
```

We experienced some problems with older versions of gcc. Also, be aware that gcc and g++-library that come with NEXTSTEP 3.2 have bugs in the Rational library. Please use gcc and g++lib on the newest version NEXTSTEP 3.3, or build a recent gcc and g++library on older systems. Generally, cdd+ seems to be most stable when compiled with gcc-2.6.3 and libg++-2.6.2.

Note that cddr+ reads Polyhedra data in integer or rational number type, while cddf+ reads data in integer or real number type.

(2) [Recompilation] The first two constants in the program cdddef.h are to be changed by the user if necessary, and the program must be recompiled each time after any change is made. These constants are simply to specify the largest size of acceptable input data $(b, -A)$:

```
#define MMAX    5001  /* USER'S CHOICE: max row size of A plus one */
#define NMAX    101   /* USER'S CHOICE: max column size of A plus one */
```

If this input data has m rows and $d+1$ columns, then in the program, MMAX should be at least $m+1$ and NMAX should be at least $d+1$. Although it has no sense to set the sizes MMAX and NMAX much larger than necessary, the program only creates spaces for MMAX+NMAX pointers and uses only necessary storage space for each input, and thus large MMAX and NMAX won't be too harmful.

Unlike the pascal version pdd, one can set the size MMAX as large as one wants. It is no more restricted by the SET TYPE element sizes of usual Pascal compilers.

(3) [TURBO/THINK C Users] Since cdd+ uses the GNU gcc libraries, it cannot be compiled with other compilers. Use the ANSI C program cdd instead which can be found in the same ftp site as cdd+.

6 Some Useful Tips for Usage

The computation is done by floating point arithmetic in cddf+ and done by rational arithmetic in cddr+. Since cddr+ runs much slower, use it when you need to make sure that the output is correct.

Clearly, there is no guarantee that the program cddf+ outputs the correct result. However cddf+ seems to work correctly for many different types of polyhedra if one carefully prepares input data files. The followings are some useful tips for input data preparation to avoid badly behaving computations with cddf+.

- In cddf+, any real value is considered as zero if its absolute value is less than 10^{-5}. Since the computation is performed with double precision arithmetic, the correctness of zero recognition depends greatly on **how accurate** the input matrix $(b - A)$ is. For example, you should never use 0.9999 for the value 1. Just use the correct value 1 as it is. Unlike many LP softwares, perturbation of data can cause some serious problems. If you want to perturb your data (e.g. right hand side) for some reason, do it with large enough constants, say of order 10^{-3}.

- If your matrix contains some irrational number, say $\sqrt{5}$, please use an approximation which is correct in at least **ten** digits, i.e. 2.236067977. See the sample input file reg600-5.ine in the ine subdirectory.

- For the same arithmetic reason, please try to scale your input matrix as even as possible by multiplying appropriate constants to some rows and columns . The program cdd does not perform any scaling before computation.

7 Bugs

- When the input system is a homogeneous system of linear inequalities (i.e. the right hand side vector b is a zero vector) and the homogeneous cone determined by the system is pointed (i.e. the origin is a vertex) , the program cdd does not output this unique vertex.

- Tope enumeration requires much storage space when the exact computation is applied. Currently we do not know how this happens although it is certain that it is something to do with Rational class library of g++. In future, this problem will be hopefully eliminated.

8 FTP site

An anonymous ftp site for the programs is set at:

```
ftpsite:  ifor13.ethz.ch (129.132.154.13)
directory: pub/fukuda/cdd
filename: cdd+-***.tar.gz
```

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

9 Other Userful Codes

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

```
 1. Geometry Center Software List:
    http://www.geom.umn.edu/software/cglist/
    (look for "arbitrary dimensional convex hull"),
  2. Linear Programming FAQ:
    http://www.skypoint.com/subscribers/ashbury/linear-programming-faq.html
    (look for "convex hull"),
  3. ZIB Berlin:
    ftp:  elib.zib-berlin.de (130.73.108.11)
    directory:  pub/mathprog/polyth.
```

Acknowledgements. I am grateful to Prof. Th. M. Liebling (EPFL, Switzerland) who provided me with an ideal opportunity to visit the department for the academic year 1993-1994. Without his generous help and encouragement, the present form of this program would not have existed. There are many people who helped me to improve cdd, in particular, I am indebted to Dr. Alain Prodon (EPFL) , Dr. Francois Margot (UBC, Canada) , Dr. Henry Crapo (INRIA, France), Dr. Alexander Bockmayr (Max Planck Institute, Germany), Mr. David Bremner (McGill University, Canada) and Mr. George Frankhauser (ETH Zurich).

Finally, but not least, I would like to thank Prof. H.-J. Lüthi (ETH Zurich) who has made the current new development (cdd+) possible. Some important new features, including the exact computation and the tope listing, have been added to cdd with his support.

References

[AF92] Avis, D. and Fukuda, K., "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra," *Discrete Comput Geometry* 8 (1992), pp. 295-313.

[CGAF93] Ceder, G., Garbulski, G.D., Avis, D. and Fukuda, K., "Ground states of a ternary lattice model with nearest and next-nearest neighbor interactions," *Physical Review B*, Vol.49, No. 1, (1994) pp. 1-7.

[G90] Grishukhin, V.P., "All facets of the cut cone for n=7 are known," European Journal of Combinatorics 11 (1990), 115-117.

[MRTT53] Motzkin, T.S. , Raiffa, H., Thompson, G.L. and Thrall, R.M., "The double description method," in "Contribution to the Theory of Games, Vol. II" (H.W. Kuhn and A.W. Tucker, eds.), Annals of Math. Studies 28, Princeton University Press, 1953, pp.81-103.