PIP/PipLib

A Solver for Parametric Integer Programming Problems
Edition 5.0, for PIP/PipLib 1.4.0
July 9th 2007

Paul Feautrier, Jean-François Collard, Cédric Bastoul

This manual is for PIP and PipLib version 1.4.0, a software which solves Parametric Integer Programming problems. That is, PIP finds the lexicographic minimum of the set of integer points which lie inside a convex polyhedron, when that polyhedron depends linearly on one or more integral parameters.

It would be quite kind to refer the following paper in any publication that results from the use of the PIP software or its library:

```
@Article{Fea88,
  author =    {P. Feautrier},
  title =     {Parametric Integer Programming},
  journal =   {RAIRO Recherche Op\'erationnelle},
  year =      1988,
  volume =    22,
  number =    3,
  pages =     {243--268}
}
```

Table of Contents

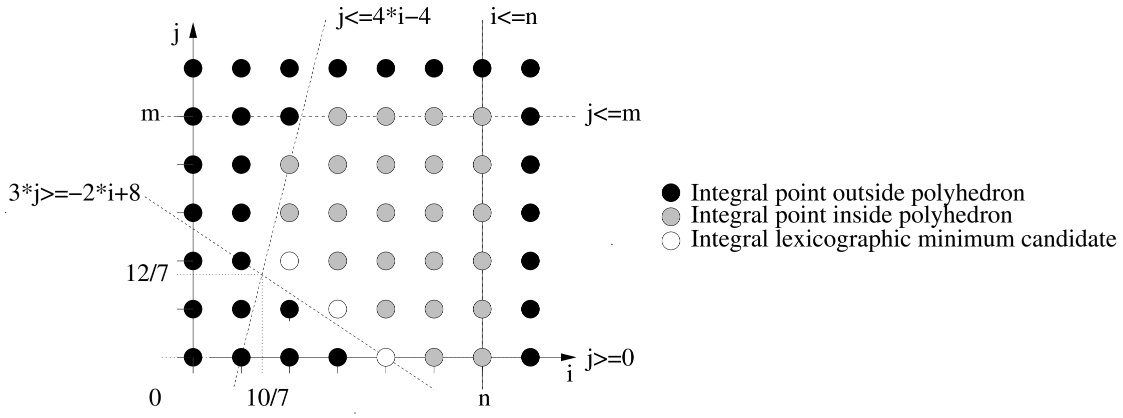
1 Introduction

PIP is a software that finds the lexicographic minimum (or maximum) in the set of integer points belonging to a convex polyhedron. The very big difference with well known integer programming tools like *lp_solve* (see [lp_solve], page 33) or *CPLEX* (see [CPLEX], page 33) is the polyhedron may depend linearly on one or more integral parameters. If the user asks for a non integral solution, PIP can give the exact solution as an integral quotient. The heart of PIP is the parameterized Gomory's Cuts algorithm followed by the parameterized Dual Simplex method (see [Fea88], page 33). The PIP Library (PipLib for short) was implemented to allow the user to call PIP directly from his programs, without file accesses or system calls. The user only needs to link his programs with C libraries.

PIP stands for *Parametric Integer Programming*. PIP/PipLib is known to be quite stable (the project began and is active since 1988) and fast. It is used in many projects, mostly but not exclusively in automatic optimizing/parallelizing compilation (e.g. to compute data dependences). PIP is a kind of *kernel* which does a simple, well defined work, but does it well. There is no plan to extend it too much. However you are very welcome and encouraged to post reports on bugs, wishes, critics, comments, suggestions or successful experiences in the forum of `http://www.PipLib.org` (preferably) or to send them to paul.feautrier@ens-lyon.fr or cedric.bastoul@inria.fr directly.

1.1 A First Basic Example

To better understand what can PIP achieve, let us consider the following 2-dimensional polyhedron where i and j are the unknown (the two dimensions of the space) and m and n are the parameters (the symbolic constants).


j<=4*i−4
i<=n
j
m
j<=m
3*j>=−2*i+8
12/7
j>=0
i
0
10/7
n
● Integral point outside polyhedron
● Integral point inside polyhedron
○ Integral lexicographic minimum candidate

It is defined by the following set of constraints:

$$\begin{cases} 2i + 3j - 8 \geq 0 \\ 4i - j - 4 \geq 0 \\ -i + n \geq 0 \\ j \geq 0 \\ -j + m \geq 0 \end{cases}$$

Classic tools to find the lexicographic minimum are challenged because of the parameters. On the contrary PIP can, without any informations on parameters, give all the solutions corresponding to all the possible cases according to the parameter values:

```
if (7*n >= 10) {
  if (7*m >= 12) {
    solution is (i = 2, j = 2)
  }
  if (2*n+3*m >= 8) {
    solution is (i = -m-(m div 2)+4, j = m)
  }
}
```

Also, it is possible to fully or partially define the parameter values. Then PIP will give only the solutions for the possible cases. For instance with the context:

$$\begin{cases} m \geq n \\ n \geq 5 \end{cases}$$

There is only one possible solution:

```
solution is (i = 2, j = 2)
```

Let the power of Parametric Integer Programming be with you.

1.2 PIP for Optimizing Compilation

The semantic analysis of programs accessing arrays often boils down to finding integer solutions to parametric linear programming problems. This is due to two main phenomena:

- Array subscripts are very often linear functions of surrounding loop counters ;
- The program's execution order enforces an order on possible solutions.

Let us consider the following example:

```
for (i = 0 ; i<= m ; i++)
  for (j = 0 ; j<= n ; j++)
    a[2*i+j] = i+j;
```

After completion of execution, for which values of k is `A[`k`]` defined, and which instances of the assignment wrote into this array element ? We can easily check that answering this question is equivalent to finding the solutions of the following system, where i, j and k are the unknowns:

$$\begin{cases} 0 \leq i \leq m \\ 0 \leq j \leq n \\ 2i + j = k \end{cases}$$

Moreover, if we want to know which instance gave its **final** value to `A[`k`]`, that is if we are looking for the **last** instance writing into `A[`k`]`, then we have to look for the maximal value of vector (i, j) according to lexicographic order. We thus consider the following polyhedron $\mathcal{F}(k, m, n)$:

$$\mathcal{F}(k, m, n) = \{< i, j > | 0 \leq i \leq m, 0 \leq j \leq n, 2i + j = k\}$$

What is the lexicographical maximum of the integer-valued vectors included in $\mathcal{F}(k,m,n)$? The aim of PIP is to solve such problems. The reader is referred to see [Fea88], page 33 for a mathematical description of the method.

1.3 General Formulation

Let $\mathcal{F}$ be a polyhedron:

$$\mathcal{F}(\vec{z}) = \{\vec{x} | \vec{x} \geq \vec{0}, A\vec{x} + B\vec{z} + \vec{c} \geq \vec{0}\}$$

In this formula, $\vec{x}$ is a vector with n entries: the vector of all unknowns. $\vec{z}$, $\vec{z} \geq \vec{0}$, is the vector built from parameters and has m entries. Polyhedron $\mathcal{F}(\vec{z})$ is a subset of $\mathbf{R}^n$ and is defined by $n+l$ inequalities: n inequalities expressing $\vec{x} \geq \vec{0}$ and the l inequalities corresponding to rows of matrix A of size $l * n$, matrix B of size $l * p$, and constant vector $\vec{c}$ of size l.

Size parameters can themselves be constrained by a set of affine inequalities $M\vec{z}+\vec{h} \geq \vec{0}$, which is called the *context* of the problem. M is an $m{*}p$ matrix and $\vec{h}$ a vector of dimension m. All data of a PIP problem: $(A, B, M, \vec{c}, \vec{h})$ are assumed to be integer-valued.

2 Using the PIP Software

2.1 A First Example

PIP takes as input a file that must be written accordingly to a grammar described in depth in a further section (see Section 2.2 [Writing the Input File], page 7). Moreover it supports some options to tune the internal algorithm or the software verbosity. They are discussed in a dedicated section (see Section 2.4 [Calling PIP], page 10). However, a basic use of PIP is not very complex and we present in this section how to compute the lexicographic minimum of a basic example discussed earlier.

The problem is to find the lexicographic minimum of a parametric 2-dimensional polyhedron where i and j are the unknown (the two dimensions of the space) and m and n are the parameters (the symbolic constants), defined by the following set of constraints:

$$\left\{\begin{array}{r} 2i+3j-8 \geq 0 \\ 4i-j-4 \geq 0 \\ -i+n \geq 0 \\ j \geq 0 \\ -j+m \geq 0 \end{array}\right.$$

We also consider a partial knowledge of the parameter values, called the *context,* expressed thanks to the following affine constraint:

$$n-3 \geq 0$$

An input file that corresponds to this problem may be the following. Note that we do not describe here precisely the structure and the components of this file (see Section 2.2 [Writing the Input File], page 7 for such information, if you feel it necessary):

```
( ( Four parts in the file:
    - comments (here !),
    - Information line: here "2 2 5 0 -1 1" meaning 2 unknown,
      2 parameters, 5 inequalities for domain, 1 ineq. for context,
      no big parameter (-1) and integer solution requested (1).
    - List of domain inequalities: #[ 2  3 -8  0  0] meaning
      (2)*i + (3)*j + (-8)*1 + (0)*m + (0)*n >= 0.
    - List of context inequalities: #[ 0  1 -3] meaning
      (0)*m + (1)*n + (-3)*1 >= 0.
  )
  2 2 5 1 -1 1
  ( #[ 2  3 -8  0  0]
    #[ 4 -1 -4  0  0]
    #[-1  0  0  0  1]
    #[ 0  1  0  0  0]
    #[ 0 -1  0  1  0]
  )
  ( #[ 0  1 -3]
    )
  )
```

This file may be called '`basic.pip`' (this example is provided in the PIP distribution as `test/basic.pip`) and we can ask PIP to process it and to print out the answer by a simple calling to PIP with this file as input: '`pip basic.pip`'. PIP will print the answer on the standard output:

```
( ( Comments are exactly the same as in input file (but not there
    for explanation purpose !). There are three major points:
    - The solution may depend on parameter values, thus it may include
      "if" constructions as "if #[condition] (then part) (else part)".
      "if #[  7  0 -12]" means "if ((7)*m + (0)*n + (-12)*1 >= 0)".
    - Final solution is either void ("()") or a list. For instance,
      "list #[  0  0  2] #[  0  0  2]" means the solution is
      i = (0)*m + (0)*n + (2)*1 and j = (0)*m + (0)*n + (2)*1.
    - New parameters representing the integer division of a parametric
      expression by a constant may be locally necessary. For instance,
      "(newparm 2 (div #[  1  0  0] 2))" means that a new parameter
      with value ((1)*m + (0)*n + (0)*1)div(2) will be present at
      index 2 in the next expressions (index starts to 0).
  )
  ( if #[  7  0 -12]
    (list #[  0  0  2]
          #[  0  0  2]
    )
    ( if #[  3  2 -8]
      (newparm 2 (div #[  1  0  0] 2))
      (list #[ -1  0 -1  4]
            #[  1  0  0  0]
      )
      ()
    )
  )
)
```

This answer in not intended to be read by humans but by computers. However it is not that difficult to translate it in a more readable way:

```
if (7*m >= 12) {
  solution is (i = 2, j = 2)
}
else {
  if (3*m+2*n >= 8) {
    solution is (i = -m-(m div 2)+4, j = m)
  }
  else {
    no solution
  }
}
```

2.2 Writing the Input File

The input text file contains a problem description, i.e. the domain, the context and few additional informations. The input text file respects the following context-free grammar (terminals are preceeded by "_"):

```
File         ::= Problem
Problem      ::= ( Comments Infos Domain Context )
Comments     ::= ( _String )
Infos        ::= Nn Np Nl Nm Bg Nq
Domain       ::= ( Vector_list )
Context      ::= ( Vector_list )
Vector       ::= #[ Integer_list ]
Vector_list  ::= Vector Vector_list | void
Integer_list ::= _Integer Integer_list | void
Nn           ::= _Integer
Np           ::= _Integer
Nl           ::= _Integer
Nm           ::= _Integer
Bg           ::= _Integer
Nq           ::= 0 | 1
```

- '`Comments`' are arbitrary strings. These comments are written verbatim to the output file, and are useful to keep track of problems and solutions.
- '`Nn`' is the number of unknowns in the program (which was denoted by n in the first section).
- '`Np`' is the number of (symbolic) parameters (p)
- '`Nl`' is the number of inequalities defining the domain of the unknowns (l).
- '`Nm`' is the number of inequalities satisfied by the parameters (m).
- '`Bg`' is the index of a *Big* parameter whose value is assumed to be infinitely large. That is, if the big parameter appears with a positive coefficient in a form ϕ, then we can immediately deduce that $\phi > 0$. If '`Bg`' is set to a nonpositive value, then there is no big parameter in the problem to be solved. Index begins at 0. Thus, since '`Bg`' is the column rank of the corresponding parameter in the '`Domain`', the first valid value for '`Bg`' is $n + 1$ (after the coefficient of the constant).
- '`Nq`' is an integer but should be interpreted as a boolean value as in C, that is, it denotes *true* if its value is nonzero. If '`Nq`' is true, then an integer-valued solution is requested. Otherwise, PIP finds the lexicographic minimum rational solution to the problem.
- '`Domain`' stores the set of inequalities defining the domain of unknowns. Each '`Vector`' represents one inequality. The entries in '`Vector`' are, in this **compulsory** order:
  - the coefficients of the unknowns (i.e., a row of matrix A),
  - the (additive) constant, (i.e., an entry of vector $\vec{c}$),
  - the coefficients of the parameters (i.e., a row of matrix B)

  This notation heavily depends on the positions given to unknowns and parameters: it is the responsibility of the user to enforce a coherent ordering of coefficients and to set a coefficient to zero when the corresponding unknown/parameter does not appear.

  There are l such '`Vector`'s in '`Domain`', and each '`Vector`' exactly has $n+1+p$ entries.

- In a similar way, '`Context`' is a list of '`Vector`'s. Each '`Vector`' represents a row of the matrix M followed by the corresponding entry in vector $\vec{h}$. '`Context`' thus includes m '`Vector`'s of $p+1$ entries.

Note that several '`Problem`'s can be given to PIP in the same file. The problems may be separated by any text that does not contain a parenthesis. By using Unix FIFOs as input and output files, it is easy to convert the present implementation of PIP into a linear programming server.

2.2.1 Example (part 1)

We consider the loop nest below (see [Fea88], page 33):

```
for (i = 0 ; i <= m ; i++)
  for (j = 0 ; j <= n ; j++)
    for (k = 0 ; k <= i+j ; k++)
      ...
```

We wish to rewrite this nest in the order k, j, i. The bounds on k can easily be guessed ($0 \leq k \leq m+n$), so let's look for the lower bound on j in the rewritten nest. This lower bound on j can be found by solving the following problem:

$$\mathcal{D}(k,m,n) = \{ <j,i> \mid i \leq m, j \leq n, k \leq i+j \}.$$

This problem is to be solved in the context $k \leq m+n$. The input file may thus look like this:

```
( ( Lower bound on j after loop inversion.
    Unknowns: j i, parameters: k m n.
  )
  2 3 3 1 -1 1
  ( #[ 0 -1  0  0  1  0]
    #[-1  0  0  0  0  1]
    #[ 1  1  0 -1  0  0]
  )
  ( #[-1  1  1  0]
  )
)
```

The first sequence of integers should be read as: this problem has 2 unknowns (i and j) and 3 parameters (k, m and n). The domain is defined by 3 inequalities, the context by 1 inequality. There is no (-1) big parameter and it is true (1) that we are looking for an integer solution.

2.3 Reading the Output File

The output file can be described by the following grammar (terminals are preceeded by "_"):

```
File            ::= Result
Result          ::= ( Comments Solution )
Comments        ::= ( _String )
Solution        ::= ( Quast_group ) | void
```

```
Quast_group      ::= Newparm_list Quast
Quast            ::= Form | (if Vector Quast_group Quast_group)
Form             ::= (list Vector_list) | ()
Newparm_list     ::= Newparm Newparm_list | void
Vector_list      ::= Vector Vector_list | void
Coefficient_list ::= Coefficient Coefficient_list | void
Newparm          ::= (newparm _Integer (div Vector _Integer))
Vector           ::= #[ Coefficient_list ]
Coefficient      ::= _Integer | _Integer / _Integer
```

The ‘`Comments`’ are copied from the input file. The ‘`Solution`’ is said to be *void* when the initial context is void. Otherwise, it is given as a quast (quasi-affine selection tree) written in a Lisp-like way. The quast may possibly be preceded by the definition of one or several new parameters (see below).

The vector coefficients may be either integers or rationals written as a division ‘`numerator/denominator`’. The latter case occurs if ‘`Nq`’ had been set to 0 in the input file.

In the solution, a ‘`Vector`’ represents an affine form; each entry is the coefficient of the corresponding parameter (the parameter of the same rank). The last entry is the additive constant.

The definition of a new parameter begins with the key-word ‘`newparm`’, then an index (rank) number, a vector of coefficients, and a denominator. The new parameter is equal to the integer division of the vector by the denominator. The new parameter can only appear in the ‘`Quast`’ following its definition. Introducing a new parameter adds one entry in the list of parameters, so the length of vectors in the solution is not constant. However, this length is always equal to 1 plus the number of original parameters plus the number of new parameters currently defined.

The solution is a multi-level conditional expression (a tree of nested conditionals.) A predicate expression p should be understood as the boolean expression $p \geq 0$. Leaves of the conditional tree are either ‘`()`’, meaning that the input problem has no solution, or a ‘`Form`’. A ‘`Form`’ is a list of vectors, each vector giving the value of the corresponding unknown.

2.3.1 Example (part 2)

The output of PIP is not intended for human consumption. No attempt has been made to implement a pretty-printer. In the interest of readability, some of the result files in this paper have been beautified by hand. The reader should not be surprised if he gets results with different layouts when running the examples.

Here is the exact output solution file for the previous example (see Section 2.2.1 [Example (part 1)], page 8):

```
( ( Lower bound on j after loop inversion.
    Unknowns: j i, parameters: k m n.
   1  )(if #[ -1 1 0 0]
(list #[ 0 0 0 0]
#[ 1 0 0 0]
)
(list #[ 1 -1 0 0]
#[ 0 1 0 0]
)
)
)
```

To express this solution, no new parameter had to be introduced. The form associated to the first conditional is $(-1)*k+(1)*m+(0)*n+(0)*1 = m-k$ so the test should be read as $m-k \geq 0$. If this inequality holds, then the solution is $< 0, k >$. Otherwise, the solution is $< k-m, m >$.

To sum things up, the lexicographical minimum of $\mathcal{D}$ is:

`if m-k >= 0 then <0, k> else <k-m, m>`.

Hence the lower bound on the first coordinate:

`if m-k >= 0 then 0 else k-m`.

2.4 Calling PIP

PIP is called by the following command:

```
pip [-s|-v...] [-d] [-z] [input [output]]
```

The default behavior of PIP is to read the input informations from the standard input and to print the solution (or some error messages if anything went wrong) on the standard output. Options and messages are discussed in next sections.

2.4.1 Options

PIP's behavior and the output shape is under the user control thanks to few options which are detailed in this section. `input` is the input file. If none is given, input is standard input. For instance, we can call PIP to process the input file `test.pip` with default options by typing: `pip test.pip` or `more test.pip | pip`. If no `output` file is given, then the results are printed to the standard output.

2.4.1.1 Verbosity

PIP prints some information on the screen after having solved a problem. The `-s` (silent mode) switches this feature off. On the contrary, the verbose `-v` option tells PIP to copy, in a file, all the input data and all the intermediary results. The name of this file is given either by the variable `DEBUG` in the environment or is built by `mkstemp`. The number of consecutive `-v`'s (from 0 to 3) controls the degree of verbosity of Pip. A word of caution: debug files may become very large very fast.

2.4.1.2 Simplification

If the `-z` option is given, then the solution is somewhat simplified. The solution of a parametric problem may be in the form of a quast all of whose leaves are nil. This means in fact that the original polyhedron is empty whatever the values of the parameters. An example, due to Dirk Fimmel, is the following:

```
( ((i j 1)(m n))
  2 2 7 0 -1 1
  ( #[2 6 -9 0 0]
    #[5 -3 0 0 0]
    #[2 -10 15 0 0]
    #[-2 6 -3 0 0]
    #[-2 -6 17 0 0]
    #[0 1 0 -1 0]
    #[1 0 0 0 -1]
  )
  ()
)
```

Without the -z option, the solution is:

```
( ((i j 1)(m n) -1 )
  ( if #[ -4 0 5]
    ( if #[ 0 -4 3]
      ()
      ( if #[ 0 -2 9]
        ( if #[ 0 -2 3]
          (newparm 2 (div #[ 0 2 3] 6))
          (newparm 3 (div #[ 0 2 10 7] 12))
          (newparm 4 (div #[ 0 4 0 2 1] 6))
          ()
          ( if #[ 0 -2 7]
            (newparm 2 (div #[ 0 4 3] 6))
            ( if #[ 0 -8 6 11]
              ()
              ()
            )
            ()
          )
        )
        ()
      )
    )
    ( if #[ -1 0 3]
      ( if #[ -1 0 2]
        ( if #[ 10 -2 -15]
          ()
          ()
        )
        ()
      )
      ()
```

```
          )
        )
      )
```

Inspection reveals that all leaves are `()`. With the `-z` option, the solution is much simpler:

```
      ( ((i j 1)(m n) -1 )
        ()
      )
```

2.4.1.3 Deepest Cut

When Pip is asked for an integral solution, it constructs new constraints (the so-called *cuts*) which eliminate fractional solutions and keep all integer solutions. The selection of cuts is somewhat arbitrary. When the `-d` option is given, Pip uses this degree of freedom to select the *deepest cut* according to an algorithm by Gondran. Intractable problems may become tractable when using this option, and conversely. Use with caution.

2.4.2 Messages

PIP may print various messages to give, e.g., some information on the complexity of the problem or to help the user to solve some errors.

2.4.2.1 General Messages

- '`Version X.x`'. Currently, `D.1`.
- '`cross : <n>, alloc : <m>`' This message is output after solving each problem. The value of `<n>` gives an idea of the complexity of the problem.

2.4.2.2 Errors Related to the Input

- '`Syntax error`': unbalanced parentheses in the input.
- '`Too much variables`'.
- '`Too much parameters`': check the input and/or change the value of constants `MAXCOL` and `MAXPARM` in file `type.h`, then rebuild PIP.
- '`Your computer doesn't have enough memory`': self explanatory.

2.4.2.3 Errors Related to the Solution

- '`Integer Overflow`': A number has been generated that is too large to be accommodated in a 32 bit integer. Check the input and/or switch to Zbigniew Chamski's infinite precision PIP.
- '`The solution is too complex`': the solution quast has grown beyond the memory allocated to it. Check the input and/or change the value of constant `SOL\_SIZE` in file `type.h`, then rebuild PIP.
- '`Memory overflow`': self explanatory.
- '`<file> unaccessible`': one of the input, output or debug file cannot be opened.

2.4.2.4 Implementation Errors

All such error messages begin by the word '`Syserr`'. These messages indicate a bug in the implementation. You should report such events by sending a copy of the input file by e-mail to the author, `Paul.Feautrier@ens-lyon.fr` who will endeavor to solve the problem as soon as possible.

2.5 The Power of PIP

In the following sections, we explain how PIP can be used to solve extended classes of problems:

- Problems where equalities occur.
- Problems where a lexicographic *maximum* has to be found.
- Cases when linear cost functions are to be optimized.
- Problems where unknowns and/or parameters may be negative.

2.5.1 Handling Equalities

When the input problem contains r affine equalities $f_i = 0$, $1 \le i \le r$, one may just write r inequalities $f_i \ge 0$ and r inequalities $f_i \le 0$, thus satisfying PIP's input syntax. However, one may notice that only $r + 1$ inequalities are needed: $f_i \ge 0$, $1 \le i \le r$, and the following inequality: $\sum_{i=1}^{r} f_i \le 0$.

2.5.2 The Bigparm Trick

In some cases, it is useful to suppose that one parameter in a PIP problem grows *very large*. Some examples will be given in the following sections. Let B be the name of this parameter. Suppose that in the solution, one of the predicates is:

$$aB + b \ge 0,$$

where b may depend on all other parameters. For B large enough, if $a > 0$ then the predicate is true, and if $a < 0$ then the predicate is false. One can find the limit shape of the solution by removing such tests and replacing them by their true of false branch, as appropriate. This can be done a posteriori on the results of PIP, or PIP can do it *on the fly* while solving the problem. This last method is more efficient, since it tends to simplify the solution.

PIP is notified of the presence of a big parameter by setting the '`Bg`' argument to a positive value. This value is the rank of the big parameter in the problem domain. Hence, the lowest admissible value for '`Bg`' is '`Nn + 1`'.

The reader should convince himself that in the presence of two big parameters, no such simplifications are possible unless one has some information on the relative size of the parameters. Such situations should be handled by giving PIP ordinary parameters, and doing the simplification on the solution in the light of extra knowledge.

2.5.3 Computing Lexicographic Maxima

To get the maximum of an unknown x, minimize $B - x$, where B is a new *big* parameter. Adding a parameter just adds one column in the problem domain. The fact that this column corresponds to a big parameter is specified by setting the 5-th switch to a positive value, this value being the position of the column of B in the problem domain (column index starts to 0).

These cases can be handled systematically in the following way. Suppose that we are asked for the integer maximum of the polyhedron:

$$\left\{\begin{array}{r}x \geq 0\\ y \geq 0\\ 3y \leq x + 12\\ y \geq 2x - 3\end{array}\right.$$

Let us introduce the new unknowns:

$$x' = B - x,\ \ y' = B - y,$$

where B is the big parameter. The system translates to:

$$\left\{\begin{array}{r}-x' + B \geq 0\\ -y' + B \geq 0\\ -x' + 3y' + 12 - 2B \geq 0\\ 2x' - y' + 3 - B \geq 0\end{array}\right.$$

Finding the maximum of $(x, y)^T$ is equivalent to finding the minimum of $(x', y')^T$, provided B is large enough. The solution of the above problem is:

```
( (a maximization problem 1)
  ( if #[ -1 6]
    ( if #[ -1 3]
      (list #[ 0 0]
            #[ 0 0]
      )
      ( if #[ -5 27]
        (newparm 1 (div #[ 1 1] 2))
        (list #[ 1 -1 -1]
              #[ 0  0  0]
        )
        (list #[ 1 -4]
              #[ 1 -5]
        )
      )
    )
    (list #[ 1 -4]
          #[ 1 -5]
    )
  )
)
```

Suppose we tell PIP that B is a large parameter. The input file is now:

```
( (a maximization problem)
  2 1 4 0 3 1
  ( #[-1  0  0  1]
    #[ 0 -1  0  1]
    #[-1  3 12 -2]
    #[ 2 -1  3 -1]
  )
  ()
)
```

and the solution is much simpler:

```
( (a maximization problem 1)
  (list #[ 1 -4]
        #[ 1 -5]
  )
)
```

The reader may care to check that this result is equivalent to the previous one as soon as $B > 5$. The position of the minimum is: $x' = B - 4$, $y' = B - 5$, from which we deduce: $x = 4, y = 5$. As expected, B has disappeared from the solution. If this does not happen, we observe first that B must have a positive coefficient in the result (if not, one of the inequalities $x, y \geq 0$ would be violated for B large enough). This means that the original polyhedron is not bounded, since, whatever B, it contains a point whose coordinates are $O(B)$, and hence has no maximum.

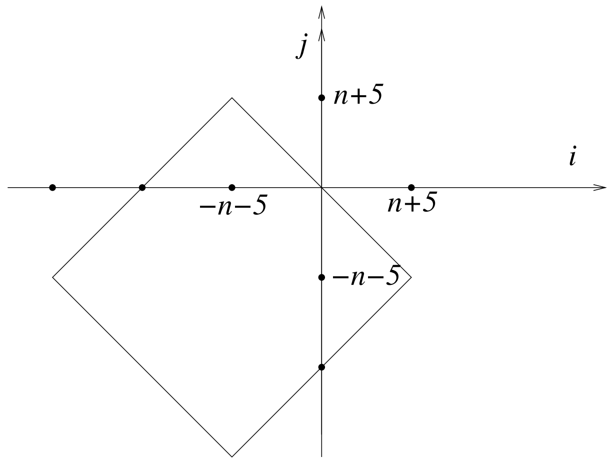
2.5.4 Optimizing Linear Cost Functions

The problem here is to compute the minimum of a linear function $\vec{c}\vec{x}$ in a polyhedron $\mathbf{P}$, where $\vec{c}$ is a vector with integer coefficients. Let us introduce a new unknown y. Solve the linear programming problem obtained by adding the constraint $y \geq \vec{c}\vec{x}$ to the defining constraints of $\mathbf{P}$. y should be the first unknown in the lexicographic ordering. Let $(y_s, \vec{x}_s)$ be the solution. Suppose that the minimum of $\vec{c}\vec{x}$ in $\mathbf{P}$ is obtained at $\vec{x}_m$ and set $y_m = \vec{c}\vec{x}_m$. Since $\vec{x}_s$ is in $\mathbf{P}$, and $y_s \geq \vec{c}\vec{x}_s$, it is clear that $y_s \geq y_m$. Conversely, $(y_m, \vec{x}_m)$ satisfies the constraints of the problem of which $(y_s, \vec{x}_s)$ is the lexicographic minimum. Hence $(y_s, \vec{x}_s) \ll (y_m, \vec{x}_m)$, and, since y is the first unknown, $y_s \leq y_m$. Hence, $y_m = y_s$. There is no guarantee, however, that $\vec{x}_s = \vec{x}_m$ (but if they differ, solutions are equally good since $y_s = \vec{c}\vec{x}_s$ and $y_m = y_s = \vec{c}\vec{x}_s = \vec{c}\vec{x}_m$).

2.5.5 Negative Unknowns and Parameters

Suppose we want to find the minimum of $f(i, j) = i - 2j$ over the square domain defined by $\{(i, j) | -4n - 20 \leq i + j \leq 0, -2n - 10 \leq i - j \leq 2n + 10\}$

(this example was proposed and solved by Pierre Boulet):



As above, we introduce a new unknown f and the inequality $f - i + 2j \geq 0$. Since we want to optimize f, f will appear as the first unknown.

To allow n (or any other parameter) to become negative, we apply the standard trick of replacing n by $n = n' - n''$, where n' and n'' are two new parameters, both non-negative. For handling possibly negative unknows, we add a number G to each of the unknowns that ensures that

$$f' = G + f$$
$$i' = G + i$$
$$j' = G + j$$

are all non-negative. That is, G should be such that

$$G \geq \max(0, -i, -j, -f).$$

Hence, G is again a big parameter. After replacement of i, j, n and f by the new variables i', j', n', n'' and f', we obtain the set

$$\begin{aligned}\{\,(f', i', j')\ |&f' - i' + 2j' - 2G \geq 0 \wedge\\ &4(n' - n'') - 20 \leq i' + j' - 2G \leq 0 \wedge\\ &-2(n' - n'') - 10 \leq i' - j' \leq 2(n' - n'') + 10\,\},\end{aligned}$$

which corresponds to the following input:

```
( ( Solving MIN(i-2.j) under the following constraints:
    Unknowns may be negative.
    Order: f' i' j' constant G n'' n'
  )
  3 3 5 0 4 1
  ( #[ 1 -1  2  0 -2  0  0 ]
    #[ 0  1  1 20 -2 -4  4 ]
    #[ 0 -1 -1  0  2  0  0 ]
    #[ 0  1 -1 10  0 -2  2 ]
    #[ 0 -1  1 10  0 -2  2 ]
  )
  ()
)
```

The result is:

```
( ( Solving MIN(i-2.j) under the following constraints:
    Unknowns may be negative.
    Order: f' i' j' constant G n'' n' -1
  )
  ( if #[ 0 -1 1 5]
    (list #[ 1  3 -3 -15]
          #[ 1  1 -1  -5]
          #[ 1 -1  1   5]
    )
    ()
  )
)
```

which should be read as:

$$\begin{aligned}(f', i', j') = {} & \texttt{if } (-n'' + n' + 5 \geq 0) \\ & \texttt{then } (G + 3n'' - 3n' - 15, G + n'' - n' - 5, G - n'' + n' + 5) \\ & \texttt{else } \perp\end{aligned}$$

That is, in the original coordinate system:

$$(f, i, j) = \texttt{if } (n \geq -5) \texttt{ then } (-3n - 15, -n - 5, n + 5) \texttt{ else } \perp$$

I.e., the minimum value for function f is $-3n - 15$, and this value is reached at point $(-n - 5, n + 5)$. This minimum exists only if $n \geq -5$; otherwise, the feasible set is empty.

2.5.6 Mixed Programming

A mixed program is a program in which some variables are constrained to be integers while others may take rational values. Suppose for instance that we have to solve:

$$\begin{aligned}S = {} & \min ax + by, \\ & Ax + By + c \geq 0,\end{aligned}$$

where y is the vector of the integer variables. First, solve

$$\begin{aligned} T = &\min ax, \\ &Ax + By + c \geq 0, \end{aligned}$$

in rational, with y as parameters. The result is a quast. To each leaf i is associated a linear function $f_i(y)$ and a set of inequalities $C_i y + d_i \geq 0$. T is equal to f_i when y is such that the corresponding inequalities are satisfied. For each i, solve the problem:

$$\begin{aligned} S_i = &\min f_i(y) + by, \\ &C_i y + d_i \geq 0, \end{aligned}$$

in integers. The final result is the minimum of all S_i. Obviously, the method can accommodate parameters in the constraints. The S_i will be functions of these parameters, and the minimum must be computed symbolically.

3 Using the PIP Library

The PIP Library (PipLib for short) was implemented to allow the user to call PIP directly from his programs, without file accesses or system calls. The user only needs to link his programs with C libraries. The PipLib mainly provides one function which takes as input the problem description and some options, and returns a `Quast` (see Section 2.3 [Reading the Output File], page 8) corresponding to the solution. Some other functions are provided for convenience reasons ; they are described in a further section (see Section 3.2 [PipLib Functions], page 22). Most of them require some specific structures to represent the problem or the solution; these structures are described in the next section (see Section 3.1 [PipLib Data Structures], page 19).

3.1 PipLib Data Structures Description

In this section, we describe the data structures used by the PIP library to represent and to process a parametric integer programming problem.

3.1.1 PipMatrix

The `PipMatrix` structure is a copy of the PolyLib `Matrix` data structure (see [Wil93], page 33, and `polylib/types.h`). This structure is devoted to represent a set of constraints. It is defined as the following:

```
struct pipmatrix
{ unsigned NbRows ;    /* Number of rows. */
  unsigned NbColumns ; /* Number of columns. */
  Entier ** p ;        /* Array of pointers to the matrix rows. */
  Entier * p_Init ;    /* Matrix rows contiguously in memory. */
  int p_Init_size ;    /* For internal use. */
}
typedef struct pipmatrix PipMatrix;
```

The whole matrix is stored in memory row after row at the `p_Init` address. `p` is an array of pointers where `p[i]` points to the first element of the i^{th} row. `NbRows` and `NbColumns` are respectively the number of rows and columns of the matrix. Each row corresponds to a constraint. The first element of each row is an equality/inequality tag. The constraint is an equality $p(x) = 0$ if the first element is 0, but it is an inequality $p(x) \geq 0$ if the first element is 1. The next elements are the unknown coefficients, followed by the parameter coefficients, then the scalar coefficient. **Please notice that the ordering of unknown and scalar coefficients is different from the input file of the PIP software** (this is due to historical reasons). For instance, in the problem we used as example (see Section 2.2.1 [Example (part 1)], page 8) the domain is defined by the following three constraints:

$$\begin{cases} -i + m & = 0 \\ -j + n & \geq 0 \\ j + i - k & \geq 0 \end{cases}$$

would be represented by the following rows:

```
# eq/in  i   j   k   m   n   cst
    0    0  -1   0   1   0    0
    1   -1   0   0   0   1    0
    1    1   1  -1   0   0    0
```

To be able to provide different precision version (PIP/PipLib supports 32 bits, 64 bits and arbitrary precision through the GMP library), the `Entier` type depends on the configuration options (it may be `long int` for 32 bits version, `long long int` for 64 bits version, and `mpz_t` for multiple precision version). The `p_Init_size` field is needed to free the memory allocated by `mpz_init` in the multiple precision release. Set this field to 0 if you are *not* using multiple precision. Set this field to the size of the `p_Init` array if you initialized it yourself and if you are using the multiple precision version.

The context is defined by one constraint:

$$-k + m + n \geq 0$$

the row corresponding to this constraint would be:

```
# eq/in  k   m   n   cst
    1   -1   1   1    0
```

`p_Init_size` is needed by the to free the memory allocated by `mpz_init` in the multiple precision release.

3.1.2 PipVector

```
struct pipvector
{ int nb_elements ;          /* Number of elements in the vector */
  Entier * the_vector ;      /* Vector of numerators */
  Entier * the_deno ;        /* Vector of denominators */
} ;
typedef struct pipvector PipVector ;
```

The `PipVector` structure represents a `Vector` as described in the ouput grammar (see Section 2.3 [Reading the Output File], page 8). `nb_elements` is the number of vector elements, `the_vector` is an array which contains the numerators of these elements and `the_deno` is an array which contains their denominators: the i^{th} element is `the_vector[i]/the_deno[i]`.

3.1.3 PipNewparm

```
struct pipnewparm
{ int rank ;                 /* Index of the newparm */
  PipVector * vector ;       /* Vector of parameter coefficients */
  Entier deno ;              /* Denominator for the whole vector */
  struct pipnewparm * next ; /* Pointer to next newparm */
} ;
typedef struct pipnewparm PipNewparm ;
```

The `PipNewparm` structure represents a `NULL` terminated linked list of `Newparm` as described in the ouput grammar (see Section 2.3 [Reading the Output File], page 8). For each `Newparm`, the rank is `rank`, the vector of coefficients is pointed by `vector`, and the denominator is `deno`. `next` is a pointer to the next `PipNewparm` structure.

3.1.4 PipList

```
struct piplist
{ PipVector * vector ;          /* Pointer to a vector */
  struct piplist * next ;       /* Pointer to next vector */
} ;
typedef struct piplist PipList ;
```

The `PipList` structure represents a `NULL` terminated linked list of `Vector` as described in the ouput grammar (see Section 2.3 [Reading the Output File], page 8). `vector` is a pointer to the vector of the current node and `next` is a pointer to the next `PipList` structure.

3.1.5 PipQuast

```
struct pipquast
{ PipNewparm * newparm ;        /* List of newparms */
  PipList * list ;              /* The solution (if no condition) */
  PipVector * condition ;       /* The condition */
  struct pipquast * next_then ; /* Quast if condition is true */
  struct pipquast * next_else ; /* Quast if condition is false */
  struct pipquast * father ;    /* Pointer to father quast */
} ;
typedef struct pipquast PipQuast ;
```

The `PipQuast` represents a `Quast` as described in the ouput grammar (see Section 2.3 [Reading the Output File], page 8). Each `Quast` has a tree structure and begins with a list of `Newparm` (field `newparm`). If the pointer `condition` is not `NULL`, the list of `Newparm` is followed by a conditional structure : if the condition pointed by `condition` is true, then the solution continues in the `Quast` pointed by `next_then`, in the `Quast` pointed by `next_else` otherwise. If the pointer `condition` is `NULL`, the list of `Newparm` is followed by a list of vectors (field `list`). For `Quast` manipulation convenience, a pointer to the father in the tree is provided (field `father`), obviously the father of the root is `NULL`.

3.1.6 PipOptions

```
struct pipoptions
{ int Nq ;                  /* 1 for integral solution, else 0 */
  int Verbose ;             /* Verbosity level (from -1 to 3) */
  int Simplify ;            /* 1 to simplify solution, else 0 */
  int Deepest_cut ;         /* 1 to use Deepest Cut algo, else 0 */
  int Maximize;             /* 0 for lexico minimum, 1 for maximum */
  int Urs_parms;            /* 0 for non-negative parms, else -1 */
  int Urs_unknowns;         /* 0 for non-negative unknowns, else -1 */
} ;
typedef struct pipoptions PipOptions ;
```

The `PipOptions` structure contains all the possible options ruling the PIP behaviour. Every `PipOptions` structure should be created and filled with the default values by the `pip_options_init` function (see Section 3.2.2 [pip_options_init], page 23) to ensure forward compatibility. Only after this, the user should modify the structure entries according to his wishes:

1. `Nq`: a boolean set to 1 if an integer solution is needed, 0 otherwise,
2. `Verbose`: a graduate value for debug informations:
   - -1: absolute silence,
   - 0: relative silence,
   - 1: information on cuts when an integer solution is required,
   - 2: information on pivots and determinants,
   - 3: information on arrays.

   Each option include the preceding one. If `Verbose` is not -1, most of the processing will be printed in a file. The file name is generated at random (by `mkstemp`) or set with environment variable DEBUG.
3. `Simplify`: a boolean set to 1 if some trivial quast simplifications are needed (recursive elimination of degenerated patterns like `if #[...] () ()`), 0 otherwise,
4. `Deepest_cut`: a boolean set to 1 if PIP has to use the deepest cut algorithm, 0 otherwise,
5. `Maximize`: a boolean set to 0 if the lexicographic minimum is requested, or to 1 for the lexicographic maximum. When trying to find the lexicographic maximum, the method used is the one presented in a previous section (see Section 2.5.3 [Computing Lexicographic Maxima], page 13): if no bigparm was set, a new (big) parameter is automatically created by adding a new column (at the last position) to the constraint system. This optional extra parameter is removed again from the output. Unbounded solutions have their `the_deno` set to zero. Note that setting this option allows for negative solutions. This may change in a future release.
6. `Urs_parms`: controls signs of parameters:
   - -1: all parameters have unrestricted sign,
   - 0: all parameters are non-negative.
7. `Urs_unknowns`: controls signs of unknowns:
   - -1: all unknowns have unrestricted sign,
   - 0: all unknowns are non-negative.

3.2 PipLib Functions

3.2.1 pip_solve

```
PipQuast * pip_solve
( PipMatrix * domain,      /* Domain where to find a solution */
  PipMatrix * context,     /* Constraints on parameters */
  int Bg,                  /* Bigparm index (-1 if no bigparm) */
  PipOptions * options     /* Options */
) ;
```

The `pip_solve` function solves a linear problem provided as input. The first three parameters describe the problem that the user wants to solve. The last parameter describe the options that the user has to set. These parameters are:

1. `domain`: a pointer to the equations and inequalities system which describe the unknown domain in the PolyLib constraints matrix shape,

2. `context`: a pointer to the equations and inequalities system satisfied by the parameter context in the PolyLib constraints matrix shape (it can be `NULL` if there is no context). **Caution: if there are parameters but no constraints on them, don't set `context` to `NULL` but to a matrix with the right column number (i.e., number of parameters + 2) and 0 rows because the PipLib uses this information to know the parameter number.**
3. `Bg`: the column rank of the bignum (first column rank is 0), or a negative value if there is no big parameter in the problem to be solved, if unsure, just set to -1,
4. `options`: a pointer to a data structure containing the options ruling the behaviour of PIP.

This function returns a pointer to a `PipQuast` structure containing the solution, it will be `NULL` if the context is `void`.

3.2.2 pip_options_init

```
PipOptions * pip_options_init(void) ;
```

The `pip_options_init` function allocates the memory space for a `PipOptions` structure and fills it with the default values:

- `Nq` = 1: an integer value is required,
- `Verbose` = 0: no debug informations,
- `Simplify` = 0: do not try to simplify solutions,
- `Deepest_cut` = 0: do not use deepest cut algorithm,
- `Maximize` = 0: compute the lexicographic minimum,
- `Urs_parms` = 0: all parameters are non-negative,
- `Urs_unknowns` = 0: all unknowns are non-negative.

We strongly recommend to use this function to create and initialize any `PipOptions` structure. In this way, if some new options appear in the future, there will be no compatibility issues.

3.2.3 pip_close

```
void pip_close(void) ;
```

The `pip_close` function frees the memory space that have been allocated for few global variables PipLib needs. This function has to be called when PipLib is no more useful in order to prevent slight memory leaks.

3.2.4 pip_matrix_alloc

```
PipMatrix * pip_matrix_alloc
( unsigned nb_rows,
  unsigned nb_columns
) ;
```

The `pip_matrix_alloc` function allocates the memory space for a `PipMatrix` structure with `nb_rows` rows and `nb_columns` columns. It fills the `Nb_Rows`, `Nb_Columns` and `p` fields and initializes the matrix entries to 0, then it returns a pointer to this structure.

3.2.5 pip_matrix_read

```
PipMatrix * pip_matrix_read(FILE *) ;
```

The `pip_matrix_read` function reads a matrix from a file. It takes as input a pointer to the file it has to read (possibly `stdin`), and returns a pointer to a `PipMatrix` structure. The input has the following syntax:

- some optional comment lines which begin with `#`,
- the row numbers and column numbers, possibly followed by comments, on a single line,
- the matrix rows, each row must be on a single line and is possibly followed by comments.

For instance, in the example problem (see Section 2.2.1 [Example (part 1)], page 8) the domain may be defined as follows

```
# This is the domain
3 7                 # 3 lines and 7 columns
1  0 -1  0  1  0  0 # -i + m >= 0
1 -1  0  0  0  1  0 # -j + n >= 0
1  1  1 -1  0  0  0 # j + i - k >= 0
```

3.2.6 Printing Functions

```
void pip_matrix_print(FILE *, PipMatrix *) ;
void pip_vector_print(FILE *, PipVector *) ;
void pip_newparm_print(FILE *, PipNewparm *, int indent) ;
void pip_list_print(FILE *, PipList *, int indent) ;
void pip_quast_print(FILE *, PipQuast *, int indent) ;
void pip_options_print(FILE *, PipOptions *) ;
```

There is a printing function for each structure of the PipLib. They all take as input a pointer to a file (possibly `stdout`) and a pointer to a structure. Some of them takes as input an indent step. They print the structure contents to the file without indent if `indent` < 0, with an indentation step of `indent` otherwise.

3.2.7 Memory Deallocation Functions

```
void pip_matrix_free(PipMatrix *) ;
void pip_vector_free(PipVector *) ;
void pip_newparm_free(PipNewparm *) ;
void pip_list_free(PipList *) ;
void pip_quast_free(PipQuast *) ;
void pip_options_free(PipOptions *) ;
```

There is a memory deallocation function for each structure of the PipLib. They free the allocated memory for the structure.

3.3 Example of Library Utilization

Here is a simple example showing how one can use the PipLib, assuming that a basic installation has been done. The following C program reads a domain and its context on the standard input then prints the solution on the standard output. Options are preselected : there is no bignum, we are looking for an integral solution without simplification and we don't want debug informations. This example is provided in the example directory of the PIP/PipLib distribution.

```
/* example.c */
# include <stdio.h>
# include <piplib/piplib64.h>

int main()
{ PipMatrix * domain, * context  ;
  PipQuast * solution ;
  PipOptions * options ;

  options = pip_options_init() ;
  domain  = pip_matrix_read(stdin) ;
  context = pip_matrix_read(stdin) ;

  solution = pip_solve(domain,context,-1,options) ;

  pip_options_free(options) ;
  pip_matrix_free(domain) ;
  pip_matrix_free(context) ;

  pip_quast_print(stdout,solution,0) ;
  pip_close() ;
  return 0 ;
}
```

The compilation command could be:

```
gcc example.c -lpiplib64 -o example
```

Supposing that the user wants to solve the example problem (see Section 2.2.1 [Example (part 1)], page 8), he will type:

```
3 7
1  0 -1  0  1  0  0
1 -1  0  0  0  1  0
1  1  1 -1  0  0  0

1 5
1 -1  1  1  0
```

And the program will answer:

```
( if #[ -1  1  0  0]
  (list #[  0  0  0  0]
        #[  1  0  0  0]
  )
  (list #[  1 -1  0  0]
        #[  0  1  0  0]
  )
)
```

4 Installing PIP

4.1 License

First of all, it would be very kind to refer the following paper in any publication that result from the use of the PIP software or its library, see [Fea88], page 33 (a bibtex entry is provided behind the title page of this manual, along with copyright notice, and in the PIP/PipLib home `http://www.PipLib.org`.

4.2 Adjusting the Precision

Pip is an all integer version of the dual simplex algorithm. As such, it has to handle integers whose size may grow exponentially as the computation proceeds. Integer overflow may occur and have to be checked. Since the hardware integer overflow exception is usually masked by the operating system or the compiler, overflow is detected by checking that a division somewhere in the algorithm, which can be proved to be exact by mathematical arguments, is indeed exact. If not, an error is reported and the computation stops.

The size of the numbers to be handled depends strongly on the size of the constraint matrix and on the size of its coefficients.

4.2.1 Bounded PIP

The precision of the integer representation in the Pip code can be adjusted at compile time by giving options to the `configure` shell script. By giving `configure` the option `--enable-llint-version` you ask for long long int version only (64 bits). It results in a 64 bits Pip called `pip64`. By giving `configure` the option `--enable-int-version` you ask for int version only. It results in a 32 bits called `pip32` and a faster running time.

4.2.2 Multiple Precision PIP

Multiple Precision Pip is built on top of the GMP library (this library is freely available at `http://www.swox.com/gmp`). Each integer in the program is represented as a list of 32 bits numbers. All computations are done exactly, and the size of the numbers increases as needed to preserve exactness. It follows that no overflow is possible. However, when the size of numbers increases, computations get slower and slower, and memory overflow may occur in extreme cases. In well behaved problems, 32 bits are enough for the initial data, the size of intermediate results first increases up to a maximum, then decreases, and 32 bits are again enough for the results. Hence, it has been possible to keep the input format and output format of Multiple Precision Pip completely compatible with the formats of the bounded precision versions.

To install Multiple Precision Pip, first install GMP according to the directions found at the above URL. Usually, the library is installed in `/usr/local/lib`, and the header files are in `/usr/local/include`. If this is not the case, you must adjust the Pip makefile by giving to the `configure` shell script the option `--with-gmp=PATH`, where `PATH` is the GMP library installation path. If GMP headers and libraries are not in the same installation path, you can be more precise by using `--with-gmp-include=PATH` and `--with-gmp-library=PATH` for GMP headers and libraries respectively.

By giving `configure` the option `--enable-mp-version` you ask for a GMP version only. It results in a multiple precision Pip called `pipMP`.

4.3 PIP Basic Installation

Once downloaded and unpacked (e.g. using the '`tar -zxvf piplib-1.4.0.tar.gz`' command), you can compile PIP by typing the following commands on the PIP's root directory:

- `./configure`
- `make`
- And as root: `make install`

Depending on the location of the GMP (and whether or not you want to use it), you may need to set the option `--with-gmp` of the configure script (e.g. '`./configure --with-gmp=/usr/local`' with a default GMP installation).

The program binaries and object files can be removed from the source code directory by typing `make clean`. To also remove the files that the `configure` script created (so you can compile the package for a different kind of computer) type `make distclean`.

Both the PIP software and PipLib library have been successfully compiled on the following systems:

- PC's under Linux, with the `gcc` compiler,
- PC's under Windows (Cygwin), with the `gcc` compiler,
- Mac's under MacOS X, with the `gcc` compiler,
- Sparc and UltraSparc Stations, with the `gcc` compiler.

4.4 Optional Features

The `configure` shell script attempts to guess correct values for various system-dependent variables and user options used during compilation. It uses those values to create the `Makefile`. Various user options are provided by the PIP's configure script. They are summarized in the following list and may be printed by typing `./configure --help` in the PIP top-level directory.

- By default, the installation directory is `/usr/local`: `make install` will install the package's files in `/usr/local/bin`, `/usr/local/lib` and `/usr/local/include`. The user can specify an installation prefix other than `/usr/local` by giving `configure` the option `--prefix=PATH`.
- By default, both PIP software and PipLib library are compiled and installed. By giving `configure` the option `--without-pip` the user disable the compilation and installation of the PIP software.

- By default, PIP is built in both 32 and 64 bits versions, and in multiple precision version if GMP is found by `configure`. To build only one version, the user may give to `configure` the options `--enable-int-version` or `--enable-llint-version` or `--enable-mp-version` to build only 32, 64 or multiple precision version respectively.
- By default, `configure` will look for the GMP library (necessary to build the multiple precision version) in standard locations. If necessary, the user can specify the GMP path by giving `configure` the option `--with-gmp=PATH`.

4.5 Uninstallation

The user can easily remove the PIP software and PipLib library from his system by typing (as root if necessary) from the PIP/PipLib top-level directory `make uninstall`.

5 Documentation

The Texinfo sources of the present document is provided in the `doc` directory. You can build it in either DVI format (by typing `texi2dvi piplib.texi`) or PDF format (by typing `texi2pdf piplib.texi`) or HTML format (by typing `makeinfo --html piplib.texi`, using `--no-split` option to generate a single HTML file) or info format (by typing `makeinfo piplib.texi`).

6 References

- [CPLEX] *http://www.ilog.com/products/cplex/*
- [Dant51] G.B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In Koopmans, T.C. (ed.), Activity analysis of production and allocation, John Wiley & Sons, New York, 339-347, 1951.
- [Fea88] Paul Feautrier. Parametric Integer Programming. Rairo Recherche Opérationnelle, 22(3):243–268, 1988.
- [Fea89] Paul Feautrier. Semantical analysis and mathematical programming; application to parallelization and vectorization. In M. Cosnard, Y. Robert, P. Quinton, and M. Raynal, editors, Workshop on Parallel and Distributed Algorithms, pages 309–320. Bonas, North Holland, 1989.
- [Fea92] P. Feautrier Some efficient solutions to the affine scheduling problem, part II: multidimensional time. International Journal of Parallel Programming, 21(6):389–420, December 1992.
- [Gom58] R. Gomory. Outline of an algorithm for integer solutions to linar programs. Bulletin of the American Mathematical Society 64:275-278, 1958.
- [Lem54] Lemke. The dual method for solving the linear programming problem. Naval Research Logistic Quarterly 22:978-981, 1954.
- [lp_solve] *http://groups.yahoo.com/group/lp_solve/*
- [Wil93] Doran K. Wilde. A library for doing polyhedral operations. Technical Report 785, IRISA, Rennes, France, 1993.