Solving Large Linear Optimization Problems with Scilab : Application to Multicommodity Problems

Héctor E. Rubio Scola

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

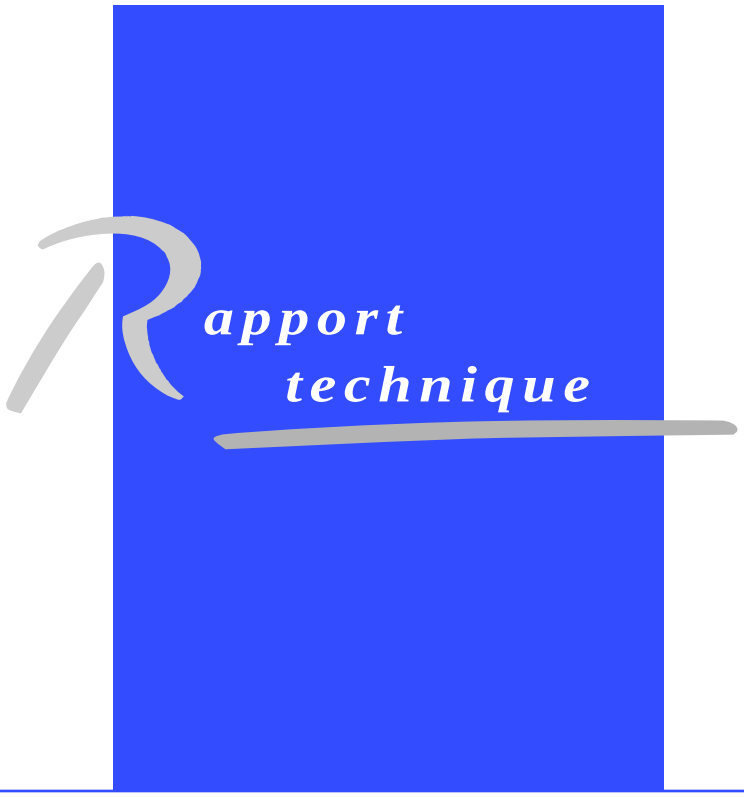
Solving large linear optimization problems with Scilab: application to multicommodity problems

Héctor E. Rubio Scola

No 0227

Janvier 1999

THÈME 4



ISSN 0249-0803

Solving large linear optimization problems with Scilab: application to multicommodity problems

Héctor E. Rubio Scola

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet Meta2

Rapport technique n ° 0227 — Janvier 1999 — 62 pages

Abstract:

We show the incorporation into the Scilab environment of the codes Lipsol, Hopdm and Lp_solve. We describe their use from Scilab and give a comparison of some results obtained. Applications for multicommodity problems are shown, which are solved in a user-friendly way using Metanet for the data introduction and visualization of the results.

Key-words: Large scale Linear programming, Interior Point methods, Mehrotra predictor-corrector algorithm, simplex algorithm, multicommodity.

(Résumé : tsvp)

CIUNR - FCEIA - Universidad Nacional de Rosario, Argentina.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : 01 39 63 55 11 - International : +33 1 39 63 55 11
Télécopie : (33) 01 39 63 53 30 - International : +33 1 39 63 53 30

Résolution de problèmes d'optimisation linéaires creux dans Scilab: application aux problèmes de multiflot

Résumé :

On décrit dans ce rapport un environnement interactif, utilisant Scilab et Metanet pour la résolution de problèmes d'optimisation linéaires creux. Trois codes sont implémentés: Lipsol, Hopdm et Lp_solve. Une application illustre l'utilisation de ces codes pour un problème de multiflot défini par Metanet.

Mots-clé : Programmation linéaire, Méthodes de Points intérieurs, algorithme prédiction-correction de Mehrotra, algorithme simplexe, multiflot

Contents

1 Introduction

Large linear optimization problems arise in many areas of operation research and control applications.

For these problems, the need for a user-friendly environment is an important issue, allowing to define the problem interactively and visualazing the results in a graphical way. The purpose of this raport is to describe such an enviroment which is based on Scilab, its graph manipulation toolbox Metanet and three numerical codes devoted to large linear optimization solvers.

Among the codes available for solving large linear optimization problems, this work has used LIPSOL[20], HOPDM[8], LP_SOLVE[17], where:

LIPSOL: is integrally written in Matlab.

HOPDM: is written in FORTRAN, only reads `.mps` files by input.

LP_SOLVE: is written in ANSI C, and with a Matlab mexfile interface.

Each of these codes has been incorporated in a different way in the Scilab systemsystem.

All of these solvers can be used by calling the respective Scilab functions, allowing the problem to be defined from the Scilab environment.

LIPSOL: was adapted to the Scilab system, without the necessity of an interface. For the calculation of the sparse Cholesky factorization, a Scilab function was created.

HOPDM: A specific interface program was written for its use from Scilab.

LP_SOLVE: A Scilab interface to interpret the Matlab mexfile was made, the sparse matrix storage was adapted, and other adaptations were carried out.

These codes may be used to solve multicommodity problems with Scilab functions of the Scilab environment. The data introduction can be done in the classic way or by the Metanet Window which allows them to be introduced in a graphic and interactive way using the mouse. The results can be visualized by colour graphics.

2 Installing the codes

2.1 Scilab Lipsol

Obtain a copy of LIPSOL from the web at
`ftp.inria.fr:`
`INRIA/Projects/Meta2/Scilab/contrib/lipsol/LIPSOL.tar.gz`
and decompress it with:
`gunzip LIPSOL.tar.gz` and `tar -xvf LIPSOL.tar`
in the subdirectory where you want to install LIPSOL. This will create the LIPSOL subdirectory structure (see "Contents" below).

Read and follow the README instructions

2.2 Scilab Hopdm

There are two web address as:

Obtain a copy of original Hopdm from the web at:
`http://www.maths.ed.ac.uk/~gondzio/software/hopdm.html`

Obtain a copy of interface Hopdm/Scilab, function Scilab and help from the web at:
`ftp.inria.fr:`
`INRIA/Projects/Meta2/Scilab/contrib/hopdm/HOPDM.tar.gz`
and decompress it with:
`gunzip HOPDM.tar.gz` and `tar -xvf HOPDM.tar`
in the subdirectory where you want to install HOPDM. This will create the HOPDM subdirectory structure (see "Contents" below).

Read and follow the README instructions.

2.3 Scilab LP_SOLVE

There are two web address as:

Obtain a copy of lp_solve with lpmex interface lp_mex Matlab from the web at:
`ftp://ftp.es.ele.tue.nl/pub/lp_solve`

Obtain a copy of function Scilab and help from the web at:
`ftp.inria.fr:`

`INRIA/Projects/Meta2/Scilab/contrib/lp_solve/LP_SOLVE.tar.gz`
and decompress it with:
`gunzip LP_SOLVE.tar.gz` and `tar -xvf LP_SOLVE.tar`
in the subdirectory where you want to install LP_SOLVE. This will create the LP_SOLVE subdirectory structure (see "Contents" below).

Read and follow the README instructions

3 The problem of solving large scale linear programs

In this report, we address the problem of solving large scale linear programs expressed in the following so-called Standard Form:

$$\begin{array}{ll} \text{minimize} & c \times x \\ \text{subject to} & A \times x \leq b \\ & x \geq 0 \end{array}$$

here $x \in \mathrm{IR}^n$ is the unknown, A is a matrix in IR^{m*n}, $m < n$, and $b \in \mathrm{IR}^m$, $c \in \mathrm{IR}^n$, $c \times x$ is the objective function, and the equations $A \times x \leq b$ are the constraints. All these entities must have consistent dimensions, of course. Usually A has more columns than rows, and $A \times x \leq b$ is therefore quite likely to be under-determined, leaving great latitude in the choice of x which minimizes $c \times x$.

3.1 What is LIPSOL?

LIPSOL (Linear programming Interior-Point SOLvers) is a package that uses Matlab's sparse-matrix data structure and MEX (Fortran-Matlab interfaces files) utilities to achieve both program simplicity and computational efficiency.It has been written in the Matlab language with Fortran-Matlab interface files by Y. Zhang [20]

This version has been written (adapted) in language Scilab [16], that uses Scilab's sparse-matrix data structure and `spchol` function (sparse cholesky factorization)

3.2 What is HOPDM?

HOPDM (Higher Order Primal-Dual Method) is a package for solving large scale linear and convex quadratic programming problems. The code is an implementation of the infeasible primal-dual interior point method. It uses multiple centrality correctors; their number is chosen appropriately for a given problem in order to reduce the overall solution time. HOPDM automatically chooses the most appropriate factorization method for a given problem (either normal equations or an augmented system). The code compares favourably with commercial LP and QP packages.

HOPDM was written by Jacek Gondzio. Its QP extension was developed together with Anna Altman.

HOPDM (Higher Order Primal-Dual Method is the library of the Fortran routines for solving large-scale linear programs with interior point methods.

The code implements a primal-dual logarithmic barrier (interior point) method with multiple correctors of centrality, i.e. the algorithm proposed by Jacek Gondzio [8].

Additionally, the code offers several useful options, e.g. a sophisticated presolve routine, a fast sparsity-exploiting Cholesky decomposition, etc., that are all carefully documented in the references.

3.3 What is LP_SOLVE?

LP_SOLVE is a Linear Programming package of the Mixed Integer Linear Program solver. It was written in the ANSI C language by Michel Berkelaar of the Eindhoven University of Technology . Most of the changes between 1.5 and 2.0 were contributed by Jeroen Dirks.

Jeffrey C. Kantor has written an LPMEX interface for the Matlab System. LPMEX is a low-level interface to the lp_solve toolkit. It may be called directly, or may be used to build higher level functions for solving various kinds of linear and mixed-integer linear programs. It uses an integer handle to point to a linear program.

An interface Scilab was made to interpret the .mex files of the Matlab System. This allows LP_SOLVE to be used directly from Scilab.

4 How to use

All the solvers can be used directly by calling the `lipsol` Scilab function, `hopdm` Scilab function, `lpsolve` Scilab function. The difference is how the file is read and how the data are used in the Scilab environment.

Let us now describe in more detail the main functions. These functions have all been implemented and have a `help` manual.

4.1 Main characteristics

characteristics	lipsol	hopdm	lp_solve
language	Scilab	Fortran	ANSI C
interface		Fortran/Scilab	Mex file Matlab
type constraints	$A \times x = b$ [1]	$b_l \leq A \times x \leq b_u$	$A \times x \leq b$
type solution	real	real	real o integer
read mps-format file and solve	no	yes	yes

4.2 How to use the solver

If the problem is defined in the Scilab environment, directly by calling the `lipsol`, `hopdm`, `lpsolve` but it exits other functions which make the process user-friendly. Depending on the algorithm chosen, the following functions are available:

LIPSOL function

Essentially, there are two functions, `lipsol` for equality constraints and `splinpro` for inequality constraints with similar syntax to the standard Scilab `linpro` function.

[1] `slinpro` acepts constraints type $A \times x \leq b$, and then it uses the `lipsol` function.

```
[x,obj] = lipsol(A,b,c,lbounds,ubounds,BIG [,fr])
[x,f] = splinpro(p,C,b,ci,cs,mi[,fr])
```

HOPDM function

There is only one function:

```
[x,obj] =
hopdm(A,b,c,lbounds,ubounds,rwstat,stavar,ranges[,param])
```

LP_SOLVE function

There are three Scilab functions, which solve the linear programme, where `lpmex` is a Scilab primitive interface function, `lpsolve` has a similar syntax `hopdm` and `lp_solve` and `lp_maker` are adapted Matlab functions.

```
[x,obj] =
lpsolve(A,b,c,rwstat [,lbounds [,ubounds [,ranges[,xint
 [,autoscale]]]]])

[obj,x,duals] =
lp_solve(f,a,b,e [,vlb [,vub [,xint [,autoscale [,keep]]]]])
```

If the primitive function `lpmex` is used, the data structure, namely `lp_handle`, has to be created before.

```
[lp_handle] =
lp_maker(f,a,b,e [,vlb [,vub [,xint [,autoscale [,setminim]]]]])

[result] = lpmex('solve', lp_handle)

[obj,x,duals] = lpmex('get_solution', lp_handle)
```

4.3 How to read MPS-format files

MPS is an old format, used to describe linear programming (see section MPS-format).

There are essentially four Scilab functions to read and to interpret this type of file:

`readmps`	-	Scilab primitive of standard distribution
`mps2lipsol`	-	Scilab function of Lipsol contribution
`mps2hopdm`	-	Scilab function of Hopdm contribution
`lpmex`	-	Scilab primitive of Lp_solve contribution

Scilab function

```
[m,n,nza,irobj,namec,nameb,namran,nambnd,name,stavar,...
rwstat,hdrwcd,lnkrw,hdclcd,lnkcl,rwnmbs,clpnts,acoeff,...
rhs,ranges,ubounds,lbounds]...
= readmps ('file-name',maxm,maxn,maxnza,big,dlobnd,dupbnd)
```

This Scilab primitive is not user friendly because it does not use the sparse storage.

LIPSOL function

```
[A,b,c,lbounds,ubounds,BIG,name] =
mps2lipsol('file-name' [,dim [,bnd]])
```

This Scilab function reads with `readmps` the `.mps` file and it transforms the original problems into a linear programming problem with equality constraints

HOPDM function

```
[A,b,c,lbounds,ubounds,rwstat,stavar,ranges,name] =
mps2hopdm('file-name' [,dim [,bnd]] )
```

This Scilab function reads equal to `mps2lipsol` but transforms the original linear programming problem with inequality constraints into problem with double equality constraints.

LP_SOLVE function

```
[lp_handle] = lpmex('read_mps', filename, verbose)
```

This Scilab function lpmex reads mps files and it stores the data in the Scilab pile. The data are accessible for the lpmex and lp_handle index.

It has to be called in this order to solve the problem:

```
[result] = lpmex('solve', lp_handle)
```

To incorporate into the Scilab environment, and to visualize the results the following has to be done:

```
[obj,x,duals] = lpmex('get_solution', lp_handle)
```

4.4 How to read MPS-format and solver

LIPSOL function

It is not possible to read directly and to solve MPS-format because `lipsol` is completly written in Scilab, consequently, the stored variables must be in the Scilab environment with sparse structure. Then, we have to use these two Scilab functions:

```
[A,b,c,lbounds,ubounds,BIG,name]
= mps2lipsol('file-name' [,dim [,bnd]])

[x,obj] = lipsol(A,b,c,lbounds,ubounds,BIG [,fr])
```

HOPDM function

```
[x,obj] = hopdm('file-name' [,dim [,bnd [,param]]])
```

LP_SOLVE function

```
[x,obj] = lpsolve('file-name'[,dim [,bnd [,param]]])
```

4.5 Example 1

We analyse now how to solve an example with the three available codes. This simple example will illustrate the syntax used for the three solvers.

Minimize

$$x_1 + 4x_2 + 9x_3$$

subject to:

$$\begin{aligned} x_1 + x_2 &\leq 5. \\ x_1 + x_3 &\geq 10. \\ -x_2 + x_3 &= 7. \end{aligned}$$

The bounds are:

$$\begin{aligned} -1 \leq x_2 &\leq 1. \\ x_1 &\leq 4. \end{aligned}$$

The corresponding `.mp` file is described in example 1, paragraph 4 (MPS-Format).

4.5.1 Using LIPSOL function

First we will read the `example1.mps` file and then we will solve, and we will analyse the other available possibilities

Read file in MPS-format

```
-->[A,b,c,lbounds,ubounds,big,name]=mps2lipsol('example1');
-->full(A)
 ans  =
!   1.    0.    1.    1.    0. !
!   0.  - 1.    1.    0.    1. !
!   0.    0.    0.  - 1.    1. !
-->b'
```

```
 ans  =
!   5.    10.    7. !
-->c'
 ans  =
!   0.    0.    1.    4.    9. !
-->lbounds'
 ans  =
!   0.    0.    0.  - 1.    0. !
-->ubounds'
 ans  =
!   1.000E+30    1.000E+30    0.    0.    1.000E+30 !
-->big
 big  =
    1.000E+30
-->name
 name  =
 EXAMPLE1
```

Here we see how `mps2lipsol` transforms the inequality constraints into equality constraints increasing the size of the variable x.

Using lipsol function

```
-->[x,obj]=lipsol(A,b,c,lbounds,ubounds,big);
-->x
 x  =
(    5,    1) sparse matrix

(    1,    1)        2.
(    3,    1)        4.
(    4,    1)      - 1.
(    5,    1)        6.
-->obj
 obj  =
    54.
```

Here we can observe the solution in x(3), x(4), x(5) because x(1) and x(2) are the auxiliary variables introduced to transform the problem.

Using splinpro function

We must transform the original problem in the following way:
all the inequalities must be $\leq$ type and the first `mi` must be equalities.

$$A = \begin{bmatrix} 0. & -1. & 1. \\ 1. & 1. & 0. \\ -1. & 0. & -1. \end{bmatrix}$$

$$b = \begin{bmatrix} 7. & 5. & -10. \end{bmatrix}'$$

because only the first constraint is an equality, we set `mi=1`;

$$c = \begin{bmatrix} 1. & 4. & 9. \end{bmatrix}'$$

$$lbounds = \begin{bmatrix} 0. & -1. & 0. \end{bmatrix}'$$

$$ubounds = \begin{bmatrix} 4. & 1. & 1.D+30 \end{bmatrix}'$$

```
[x,obj]=splinpro(c,A,b,lbounds,ubounds,mi);
-->x
 x  =
(    3,    1) sparse matrix
(    1,    1)        4.
(    2,    1)      - 1.
(    3,    1)        6.
-->obj
 obj  =
    54.
```

4.5.2 Using HOPDM function

Read file in MPS-format

```
[A,b,c,lbounds,ubounds,rwstat,stavar,ranges,name] = ...
mps2hopdm('example1');
-->full(A)
 ans  =
!   1.    1.    0. !
!   1.    0.    1. !
!   0.  - 1.    1. !
-->b'
 ans  =
!   5.    10.    7. !
-->c'
 ans  =
!   1.    4.    9. !
-->lbounds'
 ans  =
!   0.  - 1.    0. !
-->ubounds'
 ans  =
!   0.    0.    1.000E+30 !
-->rwstat'
 ans  =
!   3.    2.    1. !
-->stavar'
 ans  =
!   1.    3.    0. !
-->ranges'
 ans  =
   1.0E+29 *
!   10.    10.    10. !
-->name
```

```
 name  =
 EXAMPLE1
```

Using hopdm function

```
-->[x,obj]=hopdm(A,b,c,lbounds,ubounds,rwstat,stavar,ranges)
 obj  =
    54.
 x  =
!   4. !
! - 1. !
!   6. !
```

Read MPS-format and solve with hopdm

```
-->[x,obj]=hopdm('example1')
 obj  =
    54.
 x  =
!   4. !
! - 1. !
!   6. !
```

4.5.3 Using LP_SOLVE function

Read file MPS-format

```
[lp_handle] = lpmex('read_mps', 'example1.mps')
lp_handle  =
   0.
```

There is no possibility to restore all the variables from the `example1.mps` file into the Scilab environment, only the matrix A can be retrieved with the following commands:

```
[col_vec] = lpmex('get_column', lp_handle, col)
[row_vec] = lpmex('get_row', lp_handle, row)
[value] = lpmex('mat_elm',lp_handle,row,col)
```

It is possible to print the data from the console with

```
lpmex('print_lp', lp_handle)
```

```
problem name: EXAMPLE1
                  X         Y         Z
Minimize       1.00      4.00      9.00
    LINE1      1.00      1.00      0.00 <=      5.00
    LINE2      1.00      0.00      1.00 >=     10.00
    LINE3      0.00     -1.00      1.00  =      7.00
Type           Real      Real      Real
upbo           4.00      1.00       Inf
lowbo          0.00     -1.00      0.00
```

To load the data from a `example1.mps` file to Scilab environment, we can use `mps2hopdm`

Using lp_solve function

If the data are in the Scilab environment, we can use `lpsolve` with a similar syntax to `hopdm` or `lp_solve`.

Another possibility is to create a data base with `lp_maker` or `lpmex` and then to solve the problem using `lpmex`.

Read MPS-format and solve with lp_solve

```
[x,obj]=lpsolve('example1')
 obj  =
    54.
 x  =
!   4. !
! - 1. !
!   6. !
```

We use the Scilab function `lpmex`

```
[lp_handle] = lpmex('read_mps', 'example1.mps')
 lp_handle  =
    0.
--> [result] = lpmex('solve', lp_handle)
 result  =
    0.
-->[obj,x,duals] = lpmex('get_solution', lp_handle)
 duals  =
!   0. !
!   1. !
!   8. !
 x  =
!   4. !
! - 1. !
!   6. !
 obj  =
    54.
```

4.6 Example 2

Here we show an example in which the constraints are within a given range.

Minimize

$$x_1 + 4x_2 + 9x_3$$

subject to:

$$\begin{aligned} 4. \leq \quad & x_1 + x_2 \quad \leq 5. \\ 10. \leq \quad & x_1 + x_3 \quad \leq 15. \\ 7. \leq \quad & -x_2 + x_3 \quad \leq 9. \end{aligned}$$

The bounds are:

$$\begin{aligned} & x_1 \quad \leq 4. \\ -1 \leq \quad & x_2 \quad \leq 1. \end{aligned}$$

The corresponding `.mps` file is described in example 2, paragraph 4 (MPS-Format), and the matrix is the example 1 matrix, with the exception of the variable *ranges*

4.6.1 Using LIPSOL function

Read MPS file

Here we see how `mps2lipsol` transforms the problem into one with equality constraints

```
-->[A,b,c,lbounds,ubounds,big,name] = mps2lipsol('example2');
-->full(A)
 ans  =
!   1.    0.    0.    0.    0.    0.    1.    1.    0. !
!   0.  - 1.    0.    0.    0.    0.    1.    0.    1. !
!   0.    0.  - 1.    0.    0.    0.    0.  - 1.    1. !
!   0.    0.    0.    1.    0.    0.    1.    0.    1. !
!   0.    0.    0.    0.    1.    0.    0.  - 1.    1. !
!   0.    0.    0.    0.    0.  - 1.    1.    1.    0. !
-->b'
 ans  =
!   5.    10.    7.    15.    9.    4. !
-->c'
 ans  =
!   0.    0.    0.    0.    0.    0.    1.    4.    9. !
```

4.6.2 Using HOPDM function

Read MPS file

```
-->[A,b,c,lbounds,ubounds,rwstat,stavar,ranges,name] ...
= mps2hopdm('example2');
```

```
-->full(A)
 ans  =
!   1.    1.    0. !
!   1.    0.    1. !
!   0.  - 1.    1. !
-->b'
 ans  =
!   5.    10.    7. !
-->c'
 ans  =
!   1.    4.    9. !
-->lbounds'
 ans  =
!   0.  - 1.    0. !
-->ubounds'
 ans  =
!   0.    0.    1.000E+30 !
-->rwstat'
 ans  =
!   3.    2.    1. !
-->stavar'
 ans  =
!   1.    3.    0. !
-->ranges'
 ans  =
!   1.    5.    2. !
-->name
 name  =
 EXAMPLE2
-->[x,obj]=hopdm('example2')
 obj  =
    67.
 x  =
!   4.  !
!   0.  !
```

```
!   7.  !
```

4.6.3 Using LP_SOLVE function

```
[x,obj]=lpsolve(A,b,c,rwstat,lbounds,ubounds,ranges),
obj  =
    67.
 x  =
!   4. !
!   0. !
!   7. !
```

The same result is obtained by the following

```
[x,obj]=lpsolve('example2');
```

This function uses the `readmps` function to read the file.

```
-->[lp_handle] = lpmex('read_mps','example2.mps'),
 lp_handle  =
    0.
-->[result] = lpmex('solve', lp_handle)
 result  =
    0.
-->[obj,x,duals] = lpmex('get_solution', lp_handle)
 duals  =
!   4. !
!   9. !
!   0. !
 x  =
!   4. !
!   0. !
!   6. !
 obj  =
    58.
```

Here we obtain results that are different from the previous ones. In the `example2.mps` file, it is indicated that LINE3 is =, and now we see that `lpmex` takes it like *leq*, therefore, it does not respect the RHS-Format rules. In the Example 1, `lpmex` reads `example1.mps` correctly LINE3 with =.

```
--> lpmex('print_lp', lp_handle)

problem name: EXAMPLE2
                 X         Y         Z
Minimize      1.00      4.00      9.00
    LINE1     1.00      1.00      0.00 <=      5.00  lowbo=    1.00
    LINE2     1.00      0.00      1.00 >=     10.00  upbo=     5.00
    LINE3     0.00     -1.00      1.00 <=      7.00  lowbo=    2.00
Type          Real      Real      Real
upbo          4.00      1.00       Inf
lowbo         0.00     -1.00      0.00
```

4.7 Example 3

Minimize

$$x_1 + 4x_2 + 9x_3$$

subject to:

$$\begin{aligned} x_1 + x_2 &\leq 3. \\ x_1 + x_3 &\geq 2.2 \\ -x_2 + x_3 &= 1. \end{aligned}$$

x_1, x_2 and x_3 are integers
The bounds are:

$$\begin{aligned} x_1 &\leq 4. \\ -1 \leq x_2 &\leq 1. \end{aligned}$$

The problem is same as example 1, with the exception of the constraint and the fact that x_1, x_2 and x_3 are integers. With LIPSOL and HOPDM, it is not possible to restrict integer values.

4.7.1 Using LP_SOLVE function

To indicate that we are searching for integer solutions, we have to use the variable, variable `xint` in the following way

```
-->xint=1:3
 xint  =
!   1.    2.    3. !
-->[x,obj,t]=lpsolve(A,b,c,rwstat,lbounds,ubounds,xint)
 obj  =
  - 1.
 x  =
!   3. !
! - 1. !
!   0. !
```

Another possibility, using `lpmex`, is to read the `example1.mps`file and then modify the vector `b` and the vector `xint`

```
-->[lp_handle]=lpmex('read_mps','example1.mps')
 lp_handle  =
    0.
```

First we change the constraints

```
lpmex('set_rh_vec',lp_handle,[3 2.2 1])
 ans  =
!   0.  !
!   3.  !
!   2.2 !
!   1.  !
!   0.  !
```

then, we indicate that the three variables are integer

```
-->lpmex('set_int',lp_handle,1,1)
-->lpmex('set_int',lp_handle,2,1)
-->lpmex('set_int',lp_handle,3,1)
```

Displaying the problem, we get:

```
-->lpmex('print_lp',lp_handle)
 ans  =

problem name: EXAMPLE1
                 X         Y         Z
Minimize      1.00      4.00      9.00
    LINE1     1.00      1.00      0.00 <=      3.00
    LINE2     1.00      0.00      1.00 >=      2.20
    LINE3     0.00     -1.00      1.00  =      1.00
Type           Int       Int       Int
upbo          4.00      1.00       Inf
lowbo         0.00     -1.00      0.00
```

To solve and see the results

```
-->[result] =lpmex('solve',lp_handle);
-->[obj,x,duals] = lpmex('get_solution',lp_handle)
 duals  =
!   0. !
!   0. !
!   0. !
 x  =
!   3. !
! - 1. !
!   0. !
 obj  =
  - 1.
```

Finally, we create a `.mps` file with this new problem

```
-->lpmex('write_MPS',lp_handle,'example3.mps')
NAME          EXAMPLE1
ROWS
 N  COST
 L  LINE1
 G  LINE2
 E  LINE3
COLUMNS
    MARK0000  'MARKER'                 'INTORG'
    X         COST      1
    X         LINE1     1
    X         LINE2     1
    Y         COST      4
    Y         LINE1     1
    Y         LINE3     -1
    Z         COST      9
    Z         LINE2     1
    Z         LINE3     1
    MARK0001  'MARKER'                 'INTEND'
RHS
    RHS       LINE1     3
    RHS       LINE2     2.2
    RHS       LINE3     1
BOUNDS
 UP BND       X         4
 UP BND       Y         1
 LO BND       Y         -1
ENDATA
```

Note that if we want to read the file `example3.mps`, we find that we cannot, because the Scilab function `readmps` does not support integer constraints. A similar error message is obtained with `mps2lipsol` which also uses `readmps`.

```
-->[A,b,c,lbounds,ubounds,rwstat,stavar,ranges,name] = ...
mps2hopdm('example3')
```

```
 RDMPS1 ERROR: Unknown row found at line 8 of the MPS file.
             !--error  9999
  error MPS file
at line  21 of function mps2hopdm           called by :
[A,b,c,lbounds,ubounds,rwstat,stavar,ranges,name] =
mps2hopdm('example3')
```

5 Application: Multicommodity flow problems

5.1 Multicommodity flow problems in oriented graphs

Let G=(X,U) be an oriented graph, a non-negative capacity $b(j)$ for every edge $j = (u, v) \in$ U, and a specification $\bar{K}$ commoditities, numbered 1 through K, where the specification for commodity k consists of a source-sink pair $s^k, t^k \in$ X and a non-negative demand d_k. We will denote the number of nodes by $\bar{I}$, and the numbers of edges by $\bar{J}$.

A multicommodity flow x consists of a non-negative function $x^k(j)$ on the edges of G for every commodity k, which represents the flow of commodity k on the edge j.

The problem to be solved is to determine of the most economical way of using the available transmission capacities to route a traffic matrix through the network. This problem can be expressed as follows:

$$
\begin{array}{lllll}
min & \sum_{k,j} c_j x_j^k, & & & \\
s.t. & Ax^k & = & r^k, & k = 1, ..., \bar{K}, \\
 & \sum_k x_j^k & \leq & b_j & j = 1, ..., \bar{J}, \\
 & x_j^k & \geq & 0 & k = 1, ..., \bar{K}, j = 1, ..., \bar{J},
\end{array}
\tag{1}
$$

where:

- A is the node-arc incidence matrix corresponding to the network graph (i.e. $A_{ij} = +1$ if arc j is directed away from node i, $A_{ij} = -1$ if arc j is directed towards node i $A_{ij} = 0$ otherwise),
- k is a commodity characterized by a number of circuits, d^k, to be routed through the network between a given pair of nodes sources s^k and sinks t^k,
- r^k is the requirement vector for commodity k (i.e. $r^k_{s^k} = d^k, r^k_{t^k} = -d^k$ and $r^k_l = 0$ when $l \notin (s^k, t^k)$).
- x^k is the flow vector for a given commodity k (i.e. x^k_j is the flow on arc $j = (u, v)$ with $x^k_j > 0$)

- b_j is the capacity of link j
- c_j is the cost per unit of flow on link j.

5.2 Multicommodity flow problems in non-oriented graphs

Let H=(X,W) be a non-oriented graph, we replace each $(v,u) \in$ W, by two edges $w^+ = (v,u)$ and $w^- = (u,v)$ and we define an oriented graph , G=(X,U) with U $= W^+ \cup W^-$ where:

$W^+ = \{w^+ / w \in W\}$ and $W^- = \{w^- / w \in W\}$

Let $x^k = [x_1^{k+}; ...; x_{\bar{J}}^{k+}; x_1^{k-}, ...; x_{\bar{J}}^{k-}] = [x^{k+}; x^{k-}]$ be, where

x^k is the flow vector for a given commodity k, i.e. x_j^k is the flow on arc $j \in$ U, but U $= W^+ \cup W^-$,

x_j^{k+} is the flow on arc $j \in W^+$

x_j^{k-} is the flow on arc $j \in W^-$

Now, the problem to be solved is to determine of the most economical way of using the available transmission capacities to route a traffic matrix through the network. This problem can be expressed as follows:

$$
\begin{array}{lllll}
min & \sum_{k,j} c_j x_j^k, & & & \\
s.t. & Ax^k & = & r^k, & k = 1, ..., \bar{K}, \\
 & \sum_k (x_j^{k+} + x_j^{k-}) & \leq & b_j & j = 1, ..., \bar{J}, \\
 & x_j^{k+} & \geq & 0 & k = 1, ..., \bar{K}, j = 1, ..., \bar{J}, \\
 & x_j^{k-} & \geq & 0 & k = 1, ..., \bar{K}, j = 1, ..., \bar{J},
\end{array}
\tag{2}
$$

where:

- A is the node-arc incidence matrix corresponding to the oriented graph G.
- k is a commodity characterized by a number of circuits, d^k, to be routed through the network between a given pair of nodes sources s^k and sinks t^k,
- r^k is the requirement vector for commodity k (i.e. $r_{s^k}^k = d^k, r_{t^k}^k = -d^k$ and $r_l^k = 0$ when $l \notin (s^k, t^k)$),

- b_j is the capacity of link j
- c_j is the cost per unit of flow on link j.
- $\bar{J}$ denotes the number of edges of H (non-oriented graph) .
- $\bar{I}$ denotes the number of nodes of H equal number of arcs of G

6 Main functions

There are essentially three functions for using multiflot within Scilab, one to calculate and two to show the result.

These functions make it possible to create a completely friendly interactive graph with Metanet help, using graphic methods and the mouse. With the Metanet window we can indicate all the graph's properties like type, geometry, maximal capacities, etc.

We have to create in Scilab the three vectors of equal dimension to the different numbers of flow which contain the sources sinks and demands. Then we can see the flow results with the different colours and numbers.

These graphic results can be printed or saved in a postcript file, thanks to the graphic Scilab window.

6.1 Multiflot function

`multiflot` - multicommodity flow

CALLING SEQUENCE

```
[c,phi] = multiflot(sources,sinks,cv,g [, alg])
```

PARAMETERS

sources	:	vector, sources nodes numbers
sinks	:	vector, sinks nodes numbers
cv	:	vector, values of constrained flows
g	:	graph list
alg	:	a character string
c	:	value of cost
phi	:	matrix of the values of flows on the arcs

DESCRIPTION

`Multiflot` computes the minimum cost flow in the network g, with the value of the flow from sources nodes `sources` to sinks nodes `sinks` constrained to be equal to cv.

`Multiflot` returns the total cost of the flows on the arcs c, the matrix of the flows on the arcs phi, `phi(i,k)` = amount of commodity k circulating on edge i.

The different algorithms `'lipsol'` , `'hopdm'` (default) o `'lpsolve'` can be chosen by the optional variable `alg`.

6.2 Plot_multiflot function

`plot_multiflot` - plot of flows in the graph

CALLING SEQUENCE

```
plot_multiflot(phi,g)
```

DESCRIPTION

The function `plot_multiflot` plots graph g in a Scilab graphical window with different colours. There is a colour and a number for each flow. Each source is shown with its `cv` positive value, each sink is shown with its negative value and the saturated capacity edges are also shown. This kind of Scilab window makes it possible to print and create different types of Postcript files.

6.3 Show_multiflot function

`show_multiflot` - displays a graph and show the flows

CALLING SEQUENCE

```
show_multiflot(phi,g)
```

DESCRIPTION

The function `show_graph` displays the graph g in the current Metanet window. If there is no current Metanet window, a Metanet window is created. The return value nw is the number of the Metanet window where the graph is displayed.

This windows shows the graph in the same way as the previous one, but it cannot be printed and it is not possible to create Postcript files. Also, the graph can be modified with the mouse in an easy, interactive and friendly way.

6.4 Examples

6.4.1 Directed graphs

Example 4

We use `Metanet` to create the graph in figure 1. We can do that in a completely interactive and friendly way with the mouse. Then we choose as sources the nodes 1 and 9, as sinks the nodes 3, 7 and as capacity cv = [3, 4].

Therefore, there are two flows, one which begins from node 1 and finishes at node 9 transporting 3 and the other which begins from node 3 and finishes at node 7 transporting 4.

The data to introduce manually are:

```
sources = [1. 9.];
sinks   = [3. 7.];
cv =[ 3,4];
```

To find the solution, we use the function `multiflot` and to visualize, we use `plot_graph` and then we can print figure 2.

```
[c,phi]=multiflot(sources,sinks,cv,g)
c  =
    9.3607665
-->sparse(phi)
 ans  =
(   22,    2) sparse matrix
```

```
(    1,    1)        1.
(    3,    1)        1.
(    4,    1)        1.
(    8,    2)        2.
(    9,    1)        2.
(   14,    2)        2.
(   15,    1)        1.
(   16,    2)        2.
(   17,    1)        1.
(   19,    1)        1.
(   20,    2)        2.
(   22,    1)        2.
plot_graph(g,rep,rep1);
```

`phi(i,k)` = amount of commodity k circulating on edge i.

In figure 2, we can see in blue, with the number 1, the flow begins from node 1 and finishes at node 9. There is the green flow, with the number 2 which begins at node 3 and finishes at node 7.

Also we can see the edges with saturated capacity, namely sat. The edges which do not participate are in light yellow.

Example 5 Similar to the last example, we use `Metanet` to create the graph g which can be seen in figure 3, and in this case the source and links node are defined in the following way:

```
sources = [1, 15, 11];
sinks   = [3, 1, 1];
cv =[5, 4, 6];
```

We use, once again, `multiflot` to find the solution and `plot_graph` to see the result.

Here, as in the last case, we can see the three flows in different colours, blue (1), green (2), light blue (3) and the black edges that participate in two or more flows. We can observe the results in figure 4.

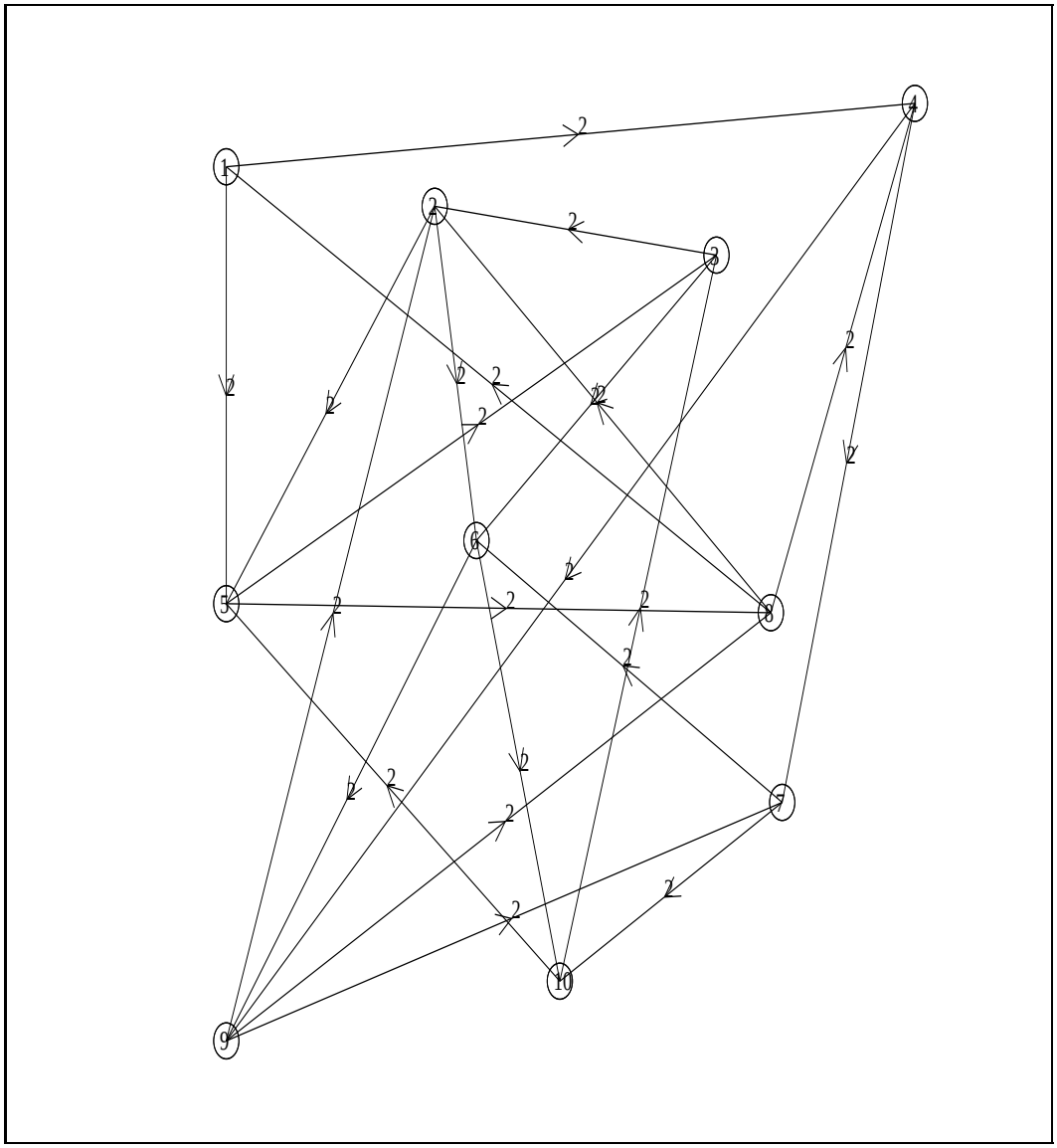


Figure 1: Example 4 original graph

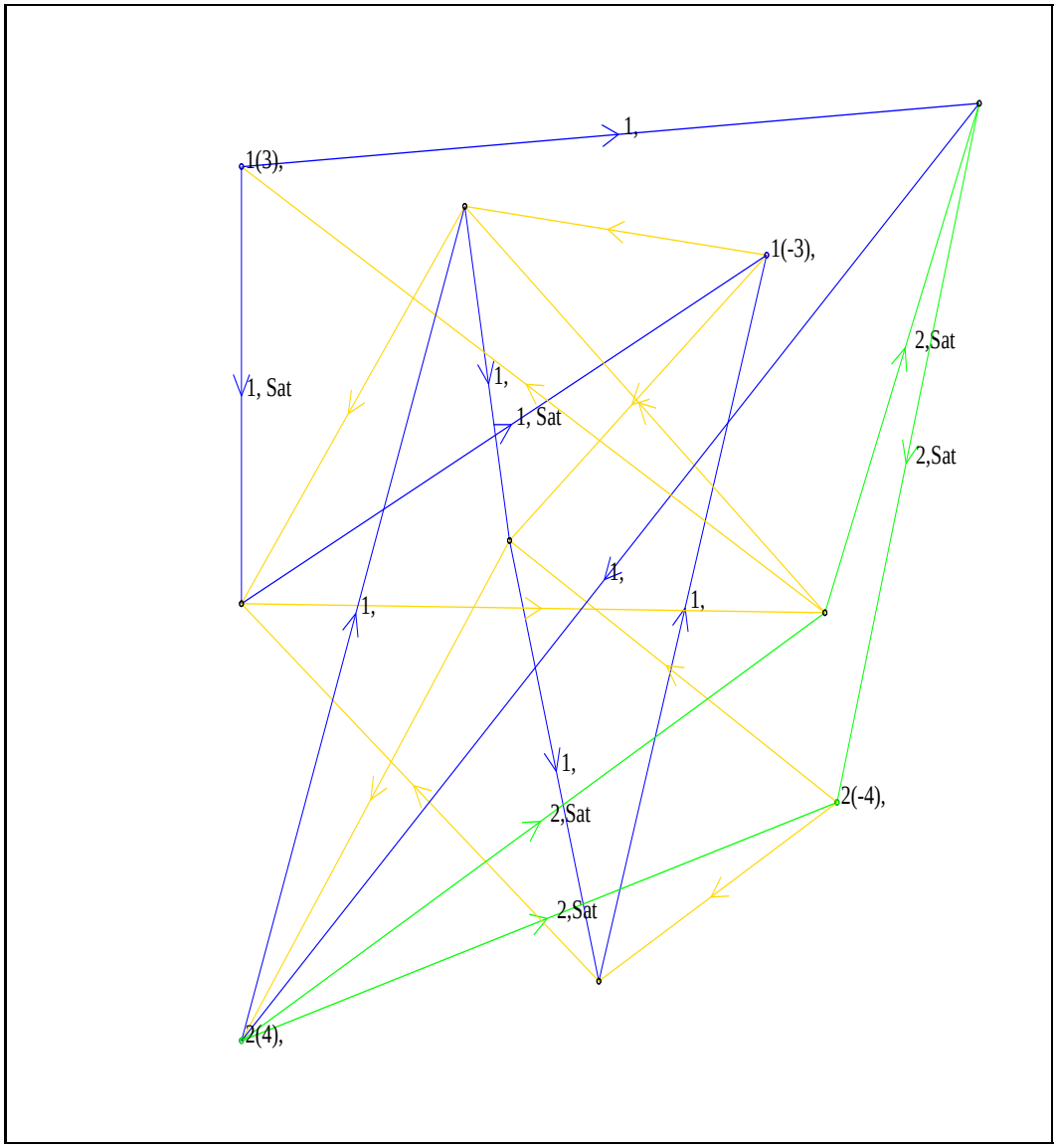


Figure 2: Example 4 solution

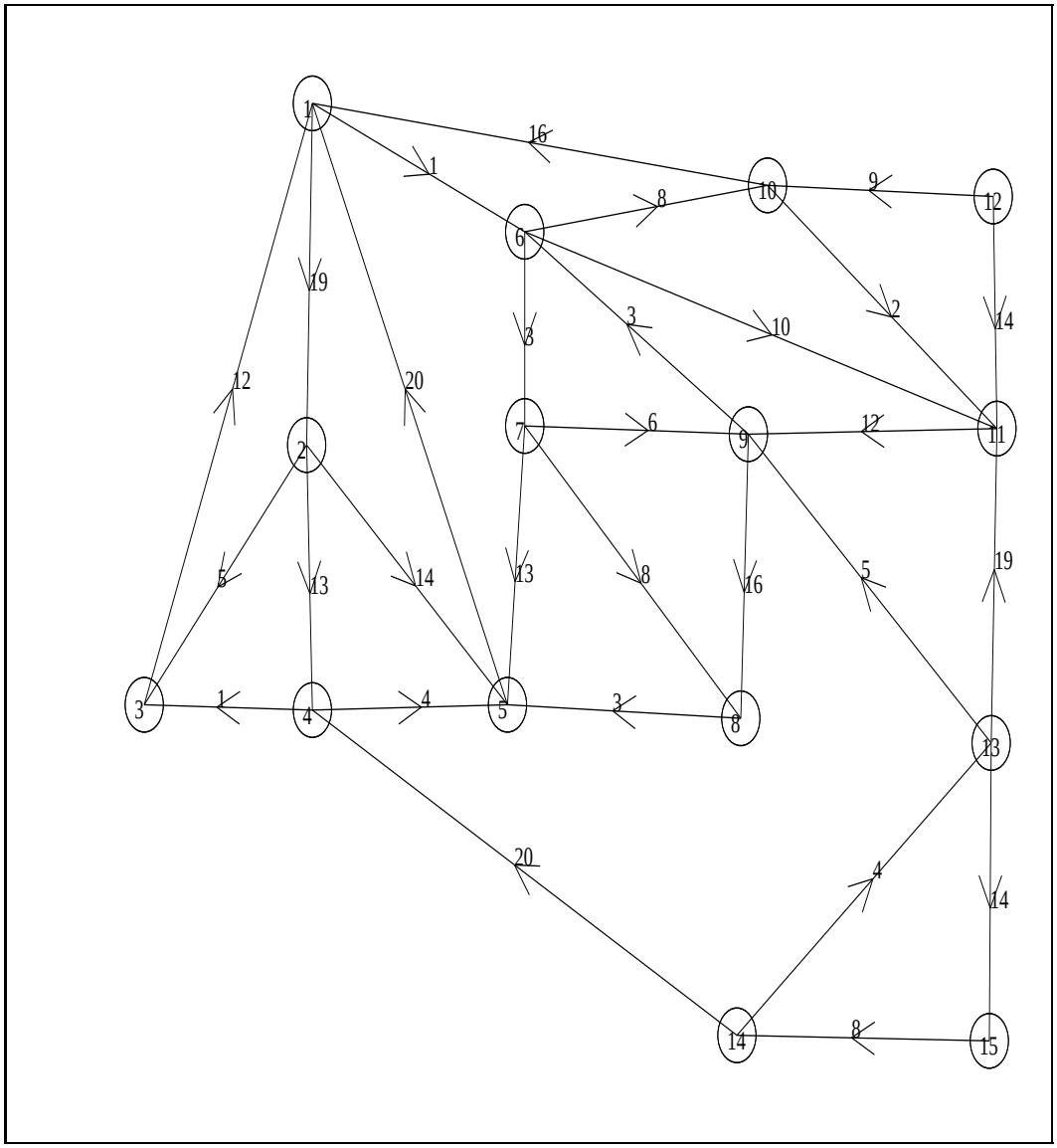


Figure 3: Example 5 original graph

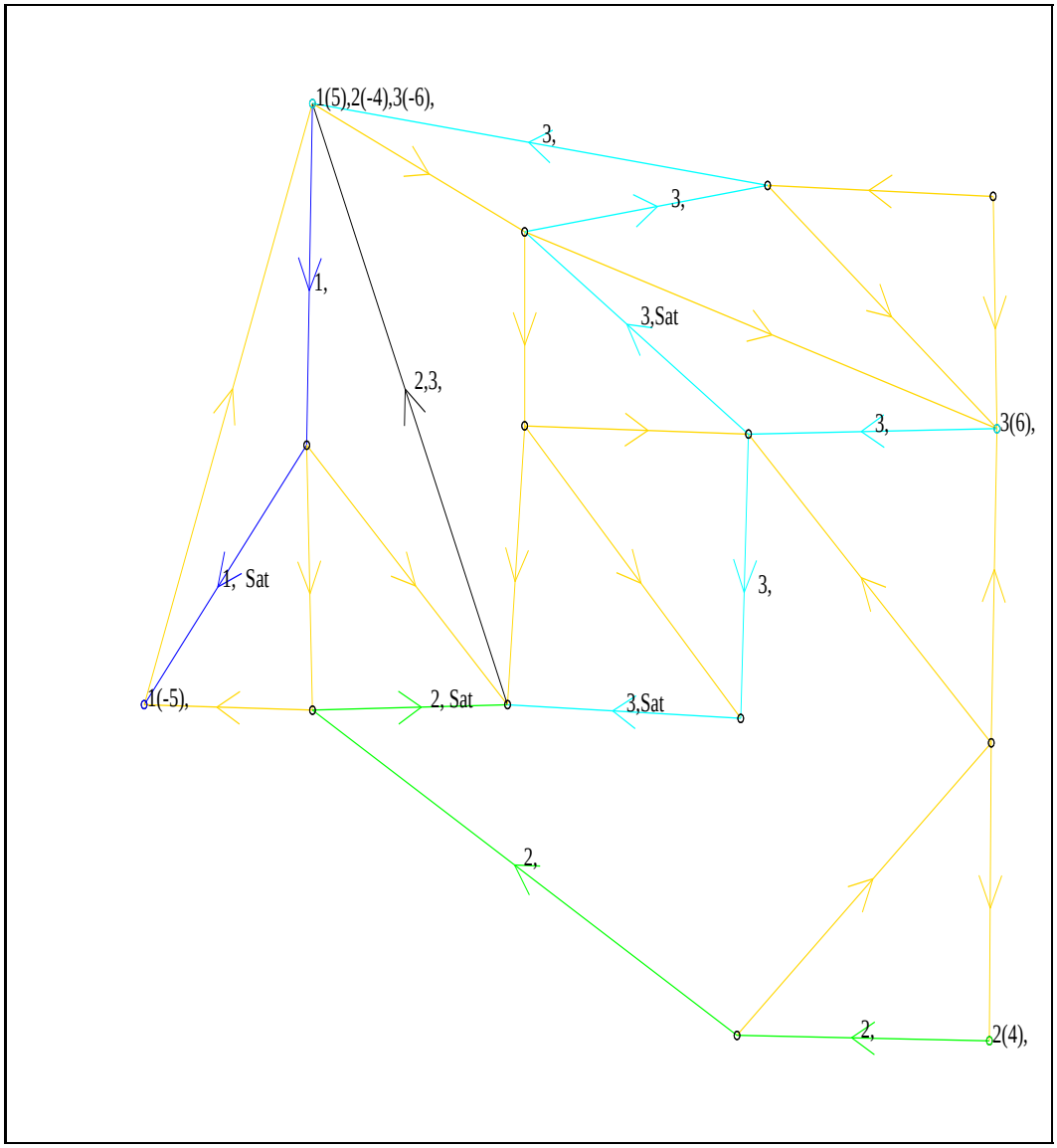


Figure 4: Example 5 solution

6.4.2 Undirected graphs

Example 6

In this example, we use the same data as in example 4, but with the difference that the graph is undirected. We can do this by modifying the `list` where we have the graph. If the graph is in `g` we do:

```
g('directed') =1
```

Once again we use, `multiflot` to find the solution and `plot_graph` to see the result.

Example 7

We proceed in the same way as in the last example, but now we use example 5 to obtain an undirected graph .

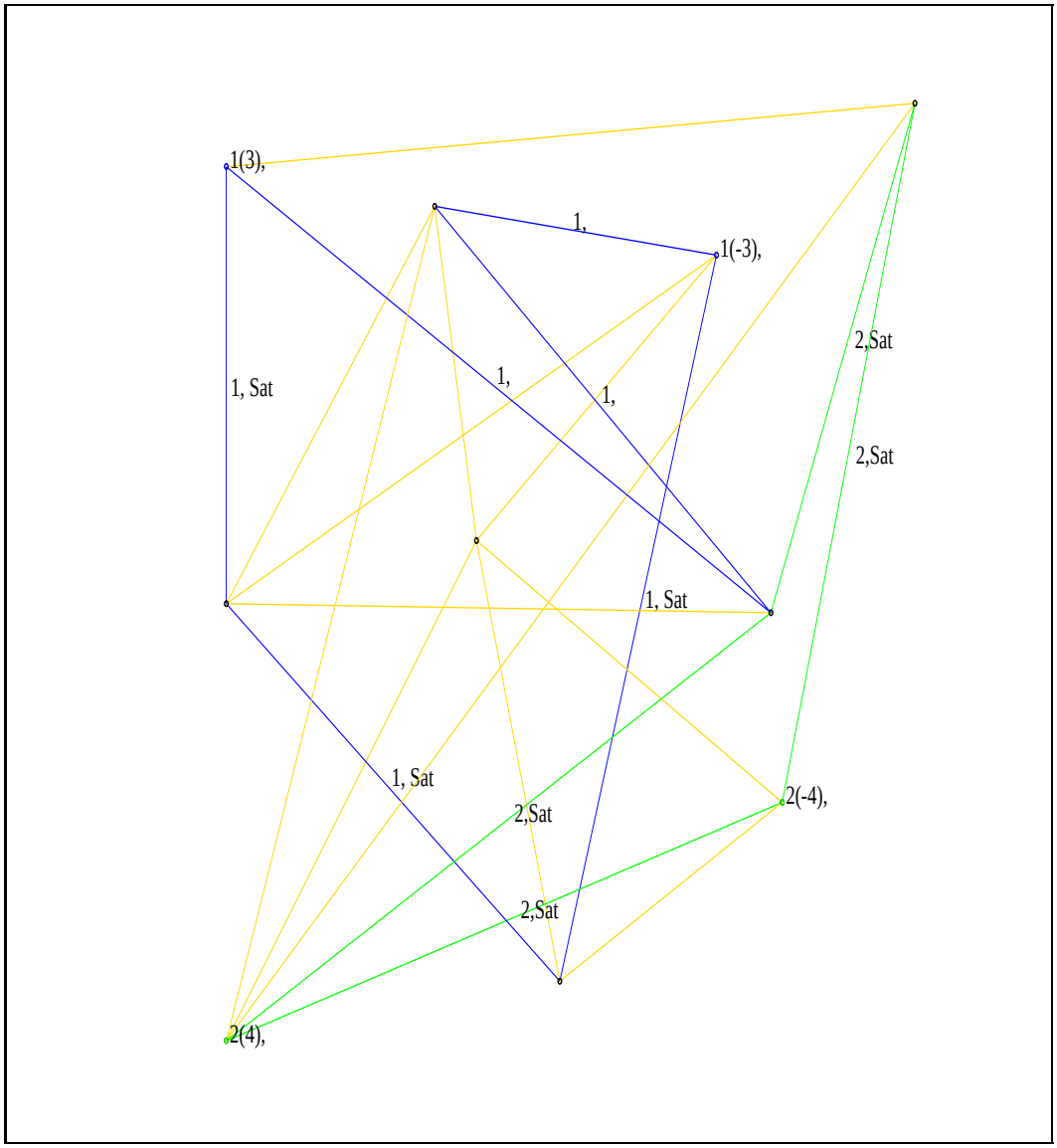


Figure 5: Example 6 undirected graph solution

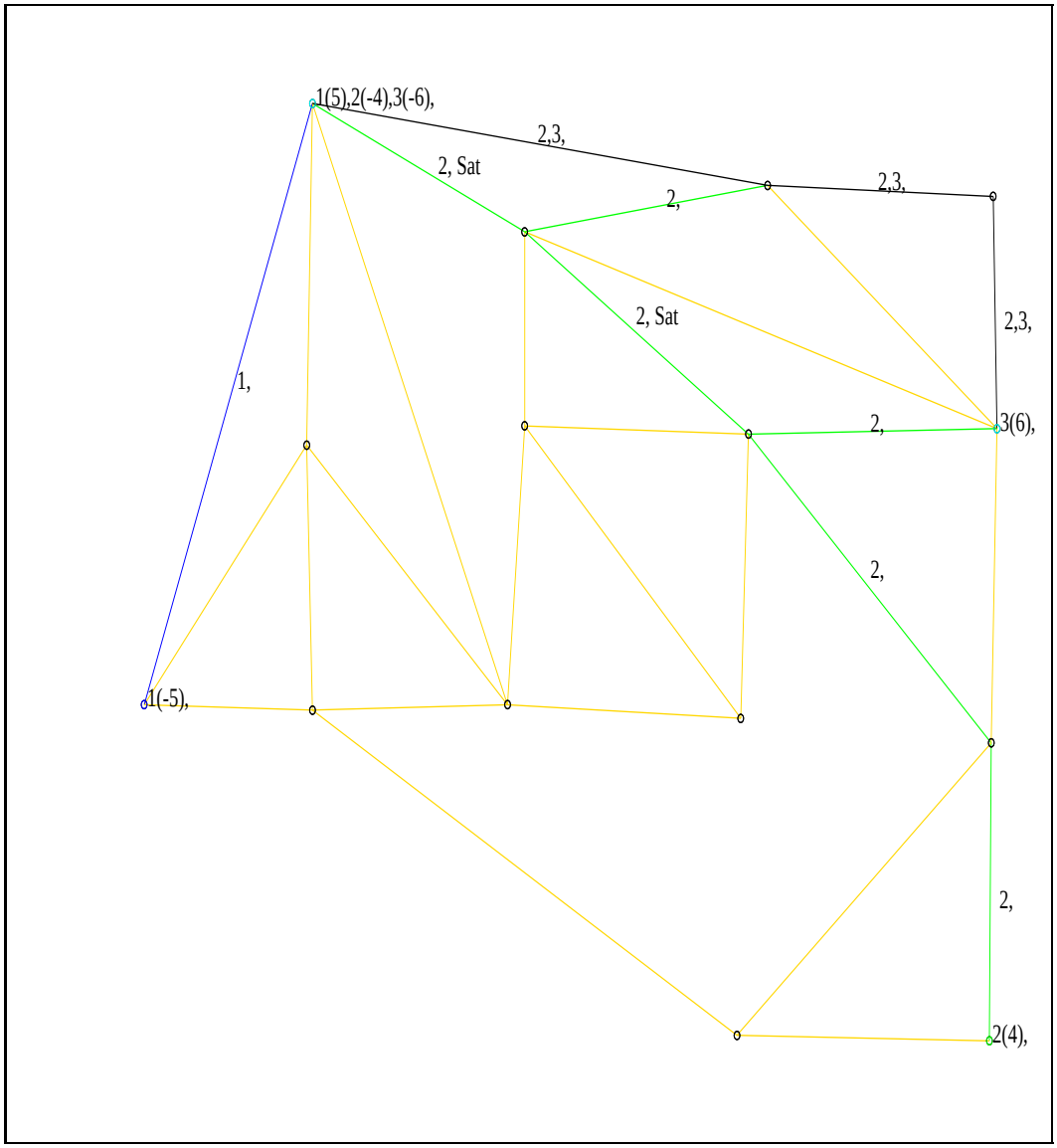


Figure 6: Example 7 undirected graph solution

7 Comparison hopdm - lipsol - lp_solve

Computational results for some problems of the NETLIB test set. This test set can be obtained on the web at

ftp://ftp.sztaki.hu/pub/oplab/LPTESTSET/NETLIB/

http://www.mcs.anl.gov/otc/Guide/TestProblems/LPtest/index.html

FILE.MPS	size	nonzeros	timer-hopdm	timer-lipsol	timer-lpsolve
ADLITTLE	[57, 97]	465	0.17	1.34	0.07
AFIRO	[28, 32]	88	0.06	0.56	0.01
AGG2	[517, 302]	4515	5.02	19.21	0.38
AGG3	[517, 302]	4531	5.25	19.15	0.38
BEACONFD	[174, 262]	3476	0.52	2.18	0.24
BLEND	[75, 83]	521	0.19	1.49	0.11
BOEING1	[351, 384]	3865	1.65	13.18	2.83
BOEING2	[167, 143]	1339	0.47	4.42	0.2
BORE3D	[234, 315]	1525	0.41	4.91	0.48
BRANDY	[221, 249]	2150	1.03	3.82	1.17
CZPROB	[930,3523]	14173	4.99	55.40	27.86
D6CUBE	[28, 32]	88	35.25	88.09	101.32
DEGEN2	[445, 534]	4449	4.36	12.43	34.46
DEGEN3	[1504,1818]	26230	91.92	147.46	1.86
E226	[224, 282]	2767	1.22	7.19	1.86
ETAMACRO	[401, 688]	2489	2.63	16.87	2.33
FINNIS	[498, 614]	2714	1.47	19.56	2.05
FIT1D	[25,1026]	14430	3.83	13.18	8.1
FIT1P	[628,1677]	10894	130.62	292.88	32.56
GANGES	[1310,1681]	7021	6.57	33.09	14.81
GFRD-PNC	[617,1092]	3467	1.12	12.79	2.
GROW7	[141, 301]	2633	0.92	4.58	0.94
KB2	[44, 41]	291	0.14	1.73	0.08
LOTFI	[154, 308]	1086	0.42	3.91	0.48
RECIPE	[92, 180]	752	0.18	1.51	0.03
SC105	[105, 103]	280	0.15	1.29	0.07
SC205	[205, 203]	551	0.29	2.00	0.81
SC50A	[50, 48]	130	0.08	0.8	0.02
SC50B	[50, 48]	118	0.07	0.72	0.02
SCAGR25	[472, 500]	2029	0.89	5.39	3.6
SCAGR7	[130, 140]	553	0.22	1.68	0.15
SCFXM1	[331, 457]	2612	1.22	7.95	1.6

SCFXM2	[661, 914]	5229	2.75	19.92	7.56
SCORPION	[389, 358]	1708	0.53	4.12	0.7
SCTAP2	[1091,1880]	2052	4.52	28.02	8.6
SCTAP3	[1481,2480]	10734	5.22	40.24	14.59
SEBA	[516,1028]	4874	0.59	119.75	1.47
SHARE1B	[118, 225]	1182	0.55	3.41	0.96
SHARE2B	[97, 79]	730	0.33	1.69	0.12
SHELL	[537,1775]	4900	2.49	11.70	3.3
SHIP04l	[403,2118]	8450	2.05	9.26	3.42
ship04s	[403,1458]	5810	1.35	6.53	2.07
SHIP12L	[1152,5427]	21597	5.62	59.81	25.35
SHIP12S	[1152,2763]	10941	2.41	22.15	10.03
SIERRA	[1228,2036]	9252	4.86	47.36	5.02
STANDMPS	[468,1075]	3038	1.81	14.65	2.07
STOCFOR1	[118, 111]	474	0.2	2.21	0.04
STOCFOR2	[2158,2031]	9492	8.28	55.10	35.13
WOODW	[1099,8405]	34478	24.86	222.49	43.69

In all the tests, we find practically the same objetive value and very similar values of `x` (solution vector). The difference we can observe is the calculation time.

In addition to the presented test, we made calculations with MIPLIB 3.0. This test set can be obtained on the web at

`http://www.caam.rice.edu/~bixby/miplib/miplib_prev.html`

We can say, for the tests made that `hopdm` and `lp_solve` are the fastest, but `hopdm` is more reliable.

8 MPS-format

MPS is an old format, used to describe linear programming. Fields start in columns 1, 5, 15, 25, 40 and 50. Sections of an MPS file are marked by so-called header lines, which are distinguished by their starting in column 1. Although it is typical to use upper-case throughout the file (as I said, MPS has long historical roots), many MPS-readers will accept mixed-case for anything except the header lines, and some allow mixed-case anywhere. The names that you choose for the individual entities (constraints or variables) are not important to the solver; you should pick names that are meaningful to you, or which will be easy for a post-processing code to read.

Example 1 The file of the problem described in Example 1 paragraph 3.3.1 Hopdm function is:

```
NAME          EXAMPLE1
ROWS
 N  COST
 L  LINE1
 G  LINE2
 E  LINE3
COLUMNS
    X         COST                 1   LINE1                1
    X         LINE2                1   LINE3                0
    Y         COST                 4   LINE1                1
    Y         LINE2                0   LINE3               -1
    Z         COST                 9   LINE1                0
    Z         LINE2                1   LINE3                1
RHS
    RHS       LINE1                5   LINE2               10
    RHS       LINE3                7
BOUNDS
 UP BND       X                    4
 LO BND       Y                   -1
```

```
 UP BND       Y                        1
ENDATA
```

For comparison, here is the same model written out in an equation- oriented format:

Optimize
COST: X + 4 * Y + 9 * Z
Subject to
LINE1: X + Y $\leq$ 5
LINE2: X + Z $\geq$ 10
LINE3: - Y + Z = 7
Bounds
0 $\leq$ X $\leq$ 4
-1 $\leq$ Y $\leq$ 1
0 $\leq$ Z
End

Strangely, there is nothing in MPS format that specifies the direction of the optimization. And there really is no standard "default" direction; some LP codes will maximize if you don't specify otherwise, others will minimize, and yet others put safety first and have no default and require you to specify it somewhere in a control program or by a calling parameter. If you have a model formulated for minimization and the code you are using insists on maximization (or vice versa), it may be easy to convert: just multiply all the coefficients in your objective function by (-1). The optimal value of the objective function will then be the negative of the true value, but the values of the variables themselves will be correct.

The NAME line can have anything you want, starting in column 15. The ROWS section defines the names of all the constraints; entries in column 2 or 3 are E for equality rows, L for less-than ($\leq$) rows, G for greater-than ($\geq$) rows, and N for non-constraining rows (the first of which would be interpreted as the objective function). The order of the rows named in this section is unimportant.

The largest part of the file is in the COLUMNS section, which is the place where the entries of the A-matrix are put. All entries for a given column must

be placed consecutively, although within a column the order of the entries (rows) is irrelevant. Rows not mentioned for a column are assumed to have a coefficient of zero.

The RHS section allows one or more right-hand-side vectors to be defined; most people don't bother to have more than one. In the above example, the name of the RHS vector is RHS1, and has non-zero values in all 3 of the constraint rows of the problem. Rows not mentioned in an RHS vector would be assumed to have a right-hand-side of zero.

The optional BOUNDS section lets you put lower and upper bounds on individual variables (no * wild lines, unfortunately), instead of having to define extra rows in the matrix. All the bounds that have a given name in column 5 are taken together as a set. Variables not mentioned in a given BOUNDS set are taken to be non-negative (lower bound zero, no upper bound). A bound of type UP means an upper bound is applied to the variable. A bound of type LO means a lower bound is applied. A bound type of FX ("fixed") means that the variable has upper and lower bounds equal to a single value. A bound type of FR ("free") means the variable has neither lower nor upper bounds.

There is another optional section called RANGES that I won't go into here. The final line must be ENDATA, and yes, it is spelled funny.

MPS input format was originally introduced by IBM to express linear and integer programs in a standard way. The format is a fixed column format, so care must be taken that all information is placed in the correct columns as described below.

The following is not intended as a complete description of MPS format, but only as a brief introduction. For more information, the reader is directed to [12].

It may be useful to look at an example MPS file while reading this MPS information.

The following template is a guide for the use of MPS format:

Field	1	2	3	4	5	6
Columns	2-3	5-12	15-22	25-36	40-47	50-61
	NAME	problem name				
	ROWS					
	type	row name				
	COLUMNS					
		column name	row name	value	row name	value
	RHS					
		rhs name	row name	value	row name	value
	RANGES					
		range name	row name	value	row name	value
	BOUNDS					
	type	bound name	col. name	value		
	ENDATA					

NAME section This section consists of just one line, with the word NAME in columns 1-4, and the title of the problem in columns 15-22.

ROWS section , each row of the constraint matrix must have a row type and a row name specified. The row type is entered in field 1 (in column 2 or 3) and the row label is entered in field 2 (columns 5-12).

The code for indicating row type is as follows:

Roow type	meaning
E	equality
L	less than or equal
G	greater than or equal
N	objective
N	no restriction (free)

COLUMNS section , the names of the variables, along with the coefficients of the objective, and all the nonzero matrix elements. The data are entered column by column, and all the data lines for the nonzero entries in each column must be grouped together contiguously. The section is preceded by a line with COLUMNS in columns 1-7, followed by data lines which may have one or two matrix elements per line.

The data line has the column label in field 2 (columns 5-12), the row label in field 3 (columns 15-22), and the value of the coefficient aij(or cj) in field 4 (columns 25-36), including a decimal point). If more than one nonzero row entry for the same column is to be made on the line, then field 5 (columns 40-47) has the next row label and field 6 (columns 50-61) has its corresponding coefficient value. It should be emphasized that the use of fields 5 and 6 is optional.

It is not necessary to specify columns for slack or surplus variables as this is taken care of automatically.

RHS section contains information for the right-hand side of the problem. The section is preceded by a line with RHS in columns 1-3. Since the right-hand side can be regarded as another column of the matrix, the data lines specifying the nonzero entries are in exactly the same format as the COLUMNS data lines, except field 2 (columns 5-12) has a label for the right-hand side. More than one right-hand side may thus be specified in this section; the one to be used for the current run is specified by its label in the agenda lines.

RANGES section(optional) is for constraints of the form: h $\leq$ constraint $\leq$ u i.e. both an upper and lower bound exist for the row. The range of the constraint is r = u - h . The value of r is specified in the RANGES section, and the value of u or h is specified in the RHS section. If b is the value entered in the RHS section, and r is the value entered in the RANGES section, then u and h are thus defined:

row type	sign of r	h lower limit	u upper limit
G	+ or -	b	b + |r|
L	+ or -	b - |r|	b
E	+	b	b + |r|
E	-	b - |r|	b

BOUNDS section (optional) , bounds on the variables are specified. When bounds are not indicated, the default bounds (0 $\leq$ x $\leq$ infinity) are assumed. The code for indicating bound type is as follows:

Field 1 (columns 2-3) specifies the type of bound:

type	meaning	
LO	lower bound	$b \leq x$
UP	upper bound	$x \geq b$
FX	fixed variable	$x = b$
FR	free variable	
MI	lower bound -inf	$-\inf \leq x$
PL	Upper bound (default)	$0 \leq x \leq +\inf$
BV	binary variable	$x = 0$ or 1

Field 2 (columns 5-12) specifies the bounds row label. More than one bounds row may be entered, but the data must be grouped together contiguously for each bounds row.

Field 3 (columns 15-22) specifies the column label corresponding to the variable

Field 4 (columns 25-36) specifies the bound value. Fields 5 and 6 are blank.

NOTE RANGES and BOUNDS are optional as are the fields 5 and 6. Everything else is required. With regard to fields 5 and 6, consider the following 2 constraints:

line1: $2x + 3y < 6$

line2: $5x + 8y < 20$

Two ways to enter the variable x in the COLUMNS section are:

Field:	2	3	4	5	6
option 1	x	LINE1	2.0	LINE2	5.0
option 2	x	LINE1	2.0		
	x	LINE2	5.0		

Section ENDATA is just one line, with ENDATA in columns 1-6, signalling the end of matrix data input.

A mixed integer program requires the specification of which variables are required to be integer. Markers are used to indicate the start and end of a group of integer variables. The start marker has its name in field 2, 'MARKER' in field 3, and 'INTORG' in field 5. The end marker has its name in field 2, 'MARKER' in field 3, and 'INTEND' in field 5. These markers are placed in the COLUMNS section.

Explained in more detail below: [12], [13].

Example 2 The file of the problem described in Example 2 paragraph 3.3.1 Hopdm function is:

```
NAME          EXAMPLE2
ROWS
 N  COST
 L  LINE1
 G  LINE2
 E  LINE3
COLUMNS
    X         COST                 1   LINE1                1
    X         LINE2                1   LINE3                0
    Y         COST                 4   LINE1                1
    Y         LINE2                0   LINE3               -1
    Z         COST                 9   LINE1                0
    Z         LINE2                1   LINE3                1
RHS
    RHS       LINE1                5   LINE2               10
    RHS       LINE3                7
RANGES
    RAN       LINE1                1
    RAN       LINE2                5
    RAN       LINE3                2
BOUNDS
 UP BND       X                    4
 LO BND       Y                   -1
 UP BND       Y                    1
ENDATA
```

9 Algorithms

9.1 LIPSOL algorithm

In this section, we outline the algorithm used in LIPSOL. For simplicity, we consider the following standard-form of linear program:

$min\{c^T x : Ax = b, x \geq 0\}$,

where $A \in \Re^{m*n}, m < n$.

Its dual linear program is

$max\{b^T y : A^T y + s = c, s \geq 0\}$.

The optimality conditions for this linear program are

$$F(z) = \begin{array}{l} Ax - b \\ A^T y + s - c \\ xs \end{array} = 0, (x, s) \geq 0,$$

where the variable $z = (x, y, s)$ contains both primal and dual variables and xs denotes the element-wise multiplication of vectors x and s.

The algorithm is a primal-dual algorithm, meaning that both the primal and the dual programs are solved simultaneously. It can be considered as a Newton-like method applied to the linear-quadratic system $F(z) = 0$ in the above optimality conditions, while keeping the iterates $\left(x^k, s^k\right)$ positive (thus the interior-point method). The algorithm can be schematically described as follows.

Let us consider just one iteration, suppressing the iteration count k for simplicity, at iterate $z = (x, y, s)$where $(x, s) > 0$. We first compute the so-called prediction direction

$\Delta z_1 = -\left[F(z)'\right]^{-1} F(z)$,

which is nothing but the Newton's direction; then the so-called corrector direction

$\Delta z_2 = -\left[F(z)\right]^{-1} \left(F\left(z + \Delta z_1\right) - \mu\ \hat{e}\right)$

where $\mu > 0$ is called the centering parameter which has to be chosen carefully, and $\hat{e}$ is a zero-one vector with ones corresponding to the nonlinear equations xs in $F(z)$. We combine the two directions with a step-length parameter α and update z to obtain the new iteration

$z^+ = z + \alpha\left(\Delta z_1 + \Delta_{Z2}\right)$

where the step-length parameter α is chosen so that $z^+ = (x^+, y^+, s^+)$ satisfies $(x^+, s^+) > 0$. The algorithm then repeats these steps at each new iteration until convergence.

The above algorithmic scheme is a variant of the predictor-corrector algorithm proposed by Mehrotra. Conceptually, it is a relatively simple algorithm though an actual implementation can be much more complicated, especially for large sparse problems. Mehrotra's approach uses three directions at each iteration: the predictor, the corrector and the centering directions. The first two directions were motivated by considering a certain trajectory linking the current iteration to a solution. The predictor is the first derivative of the trajectory and the corrector the second.

For more details on Mehrotra's predictor-corrector algorithm, its implementations and properties, see [10], [11] and [20].

9.2 HOPDM algorithm

The algorithm implemented in a version 2.12 of HOPDM is a new variant [8] of a primal-dual method proposed in [7].

9.2.1 Basic primal-dual algorithm

In this section, we outline the algorithm used in hopdm. For simplicity, we consider the following standard-form of linear program:

$$min\{c^T x : Ax = b, x \geq 0\}, \tag{3}$$

where $A \in \mathrm{IR}^{m*n}, m < n$.

Its dual linear program is

$$max\{b^T y : A^T y + s = c, s \geq 0\}. \tag{4}$$

The optimality conditions for this linear program are

$$F(x,y,s) = \begin{array}{l} Ax - b \\ A^T y + s - c \\ xs \end{array} \begin{array}{l} \\ = 0, (x,s) \geq 0, \\ \\ \end{array} \tag{5}$$

where the variables (x, y, s) contain both primal and dual variables and $x\ s$ denotes the element-wise multiplication of vectors x and s.

Next, let us replace the nonnegativity of constraints in the primal formulation with logarithmic barrier penalty terms in the objective function. This gives the following logarithmic barrier function

$$L(x, s, \mu) = c^T x - \mu \sum_{j=1}^{n} \ln x_j - \mu \sum_{j=1}^{n} \ln s_j \tag{6}$$

the first-order necessary conditions for a stationary point of 6 are

$$\begin{array}{rcl} Ax & = & b \\ x + s & = & u, \\ A^T y - w + z & = & c \\ XZe & = & \mu e \\ SWe & = & \mu e \end{array} \tag{7}$$

where X, Z, S and W are diagonal matrices with the elements $x_j, z_j, s_j,$ and w_j,respectively, and $e \in \mathrm{IR}^n$ of all ones, $z = \mu X^{-1} e$, and μ is a barrier parameter.

The algorithm is a primal-dual algorithm, meaning that both the primal and the dual programs are solved simultaneously. It can be considered as a Newton-like method applied to the linear-quadratic system 7. The algorithm can be schematically described as follows.

Having an $x, s, z, w \in \mathrm{IR}^n_+$ and $y \in \mathrm{IR}^m$, Newton's direction is obtained by solving the following system of linear equations

$$\begin{bmatrix} A & 0 & 0 & 0 & 0 \\ I & 0 & I & 0 & 0 \\ 0 & A^T & 0 & I & -I \\ Z & 0 & 0 & X & 0 \\ 0 & 0 & W & 0 & S \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \\ \Delta z \\ \Delta w \end{bmatrix} = \begin{bmatrix} b - Ax \\ u - x - s \\ c - A^T y - z + w \\ \nu e - \Delta X \Delta Z e \\ \nu e - \Delta S \Delta W e \end{bmatrix} \tag{8}$$

denote the violations of the primal and dual constraints, respectively.

The set of linear equations 8 reduces to the augmented system

$$\begin{bmatrix} -\Theta^{-1} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r \\ h \end{bmatrix} \tag{9}$$

or further to the normal equation system

$$A\Theta A^T \Delta y = A\Theta r + h \tag{10}$$

$$\begin{array}{rcl} \Theta & = & (X^{-1}Z + S^{-1}W)^{-1} \\ r & = & c - A^T y - z + w - X^{-1}(\mu e - XZe) + \\ & & +S^{-1}(\mu e - SWe) - S^{-1}W(u - x - s) \\ h & = & b - Ax \end{array} \tag{11}$$

Computing $(\Delta x, \Delta y)$ from 9 or Δy from 10 is usually divided into two phases: factorization of the matrix to some easily invertible form followed by the solution that exploits this factorization.

Once direction $(\Delta x, \Delta y, \Delta s, \Delta z, \Delta w)$ has been computed, the maximum stepsizes α_P and α_D that maintain nonnegativity of variables in the primal and dual spaces are found. Next, a new iteration is computed using a step reduction factor $\alpha_P = 0.99995$

$$\begin{array}{rcl} x^{k+1} & = & x^k + \alpha_0 \alpha_P \Delta x \\ s^{k+1} & = & s^k + \alpha_0 \alpha_P \Delta s \\ y^{k+1} & = & y^k + \alpha_0 \alpha_D \Delta y \\ z^{k+1} & = & z^k + \alpha_0 \alpha_D \Delta z \\ w^{k+1} & = & w^k + \alpha_0 \alpha_D \Delta w \end{array} \tag{12}$$

After making the step, the barrier parameter μ is updated and the process is repeated.

9.2.2 Predictor corrector technique

Factorization of the matrix 9 and 10 is usually at least one order of magnitude more expensive than the solution. The factorizations of systems 8 often take 60% to 90% of total cpu time needed to solve a problem. It is thus natural to look for a possibility of reducing their number to the necessary minimum, even at the expense of some increase in cost of a single iteration.

The predictor corrector technique proposed by Methrotha [11] decomposes a direction vector

$\Delta = (\Delta x, \Delta s, \Delta y, \Delta z, \Delta w)$ into two parts

$$\Delta = \Delta_\alpha + \Delta_c \tag{13}$$

where Δ_α and Δ_c denote affine scaling and centering components, respectively. The term Δ_α is obtained by solving 8 with $\mu = 0$ and Δ_c is the solution of equation similar to 8 with the right-hand side vector

$(0, 0, 0, \mu e - XZe, \mu e - SWe)^T$ where $\mu > 0$ is some centering parameter. The term Δ_α is responsible for optimization while Δ_c keeps the current iteration away from the boundary.

Let us observe that the affine scaling (predictor) direction Δ_α solves the linear system 8 for the right-hand side equal to the current violation of the first order optimality condition for 3 and 4, i.e. $\mu = 0$. This direction is usually "too optimistic" if a full step of length one could be made in it, the LP problem would be solved in one step. The predictor corrector makes a hypothetical step in this direction. The maximum stepsizes in the primal $\alpha_{P\alpha}$ and in the dual, $\alpha_{D\alpha}$ spaces preserving nonnegativity of (x, s) and (z, w), respectively are determined and the predicted complementarity gap.

$g_\alpha = (x + \alpha_{P\alpha}\Delta x)^T(z + \alpha_{D\alpha}\Delta z) + (s + \alpha_{P\alpha}\Delta s)^T(w + \alpha_{D\alpha}\Delta w)$ is computed. It is then used to determine the barrier parameter

$$\mu = (\frac{g_a}{g})^2 \frac{g_a}{n} \tag{14}$$

where $g = x^T z + s^T w$ denotes current complementarity gap.

For such a value of μ, the corrector direction Δ_c is computed

$$\begin{bmatrix} A & 0 & 0 & 0 & 0 \\ I & 0 & I & 0 & 0 \\ 0 & A^T & 0 & I & -I \\ Z & 0 & 0 & X & 0 \\ 0 & 0 & W & 0 & S \end{bmatrix} \begin{bmatrix} \Delta_c x \\ \Delta_c y \\ \Delta_c s \\ \Delta_c z \\ \Delta_c w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \mu e - \Delta X_a \Delta Z_a e \\ \mu e - \Delta S_a \Delta W_a e \end{bmatrix} \tag{15}$$

and, finally, the direction Δ of 13 is determined.

9.2.3 Multiple centrality correctors

Although the theory requires that subsequent iterations are in the neighborhood of the central path, in practice, they may stay quite far away from

it without negative consequences for the ability of making large steps (and fast convergence). What is really bad for a primal dual algorithm is a large discrepancy between complementary products $x_j x_j$ and $s_j w_j$.

The set Δ of 8 aims at drawing all complementary products to the same value μ. Usually, there is little hope of reaching suching a goal. The approach proposed in [7] combines the choice of targets [9] that are supposed to be easier to reach with the use of multiple correctors.

Below, this approach is presented in more detail.

Assume (x, s) and (z, w) are primal and dual solutions at a given iteration of the primal dual algorithm (x, s, z, w) are strictly positive. Next, assume that a predictor direction Δ_p at this point is determined and the maximum stepsizes in primal, α_P and dual, α_D spaces are computed with preserved nonnegativity of primal and dual variables, respectively.

The corrector direction Δ_m is looked for such that larger stepsizes in primal and dual spaces are allowed for a composite direction

$$\Delta = \Delta_p + \Delta_m \tag{16}$$

To enlarge these stepsizes from α_P and α_D to $\tilde{\alpha_P} = min(\alpha_P + \delta_\alpha, 1)$ and $\tilde{\alpha_D} = min(\alpha_D + \delta_\alpha, 1)$, respectively, a corrector term Δ_m has to compensate for the negative components in the primal and dual variables

$$\begin{array}{rcl} (\tilde{x}, \tilde{s}) & = & (x, s) + \tilde{\alpha_P}(\alpha_p x, \alpha_p s) \\ (\tilde{x}, \tilde{z}, \tilde{w}) & = & (x, z, w) + \tilde{\alpha_D}(\alpha_p y, \alpha_p z, \alpha_p w) \end{array} \tag{17}$$

The goal is supposed to be reached by adding the corrector term Δ_m that drives from this exterior trial point to the next iteration $(\hat{x}, \hat{s}, \hat{y}, \hat{z}, \hat{w})$ lying in the vicinity of central path. However, there is little chance of attaining the analytic center in one step, i.e., reaching the point

$$v = (\mu e, \mu e) \in \mathrm{IR}^{2n} \tag{18}$$

in the space of complementarity products. Hence the complementarity products of the trial point are computed

$$\tilde{v} = (\tilde{X}\tilde{z}, \tilde{S}\tilde{w}) \in \mathrm{IR}^{2n} \tag{19}$$

and concentrate effort only on correcting their outliers. Thus, these complementarity products componentwise are projected on a hypercube $H = [\beta_{min}\mu, \beta_{max}\mu]^{2n}$. This gives the following target

$$v_t = \pi(\tilde{v} \mid H) \in \mathrm{IR}^{2n} \tag{20}$$

and the following definition of the direction corrector term

$$\begin{bmatrix} A & 0 & 0 & 0 & 0 \\ I & 0 & I & 0 & 0 \\ 0 & A^T & 0 & I & -I \\ Z & 0 & 0 & X & 0 \\ 0 & 0 & W & 0 & S \end{bmatrix} \begin{bmatrix} \Delta_m x \\ \Delta_m y \\ \Delta_m s \\ \Delta_m z \\ \Delta_m w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ v_t - \tilde{v} \end{bmatrix} \tag{21}$$

Note that the right-hand side in the above system of equations has nonzero elements only in a subset of positions of $v_t - \tilde{v}$ that refer to the complementary products which do not belong to $(\beta_{min}\mu, \beta_{max}\mu)$.

Once corrector term Δ_m has been computed, the new stepsizes $\hat{\alpha_P}$ and $\hat{\alpha_D}$ are determined for the composite direction

$$\Delta = \Delta_p + \Delta_m \tag{22}$$

and the primal dual algorithm can move to the next iteration.

Quite often, the effort of factorizing $A\Theta A^T$ matrix is tens or even hundreds of times larger than that required in the following solves.

Whenever this is case, the correcting process is repeated a desireable number of times. Once advantage of the approach is that computing every single corrector term needs exactly the same effort (it is dominated by the solution of the system of equation 21).

Certain questions arise about the choice of "optimal" number of corrections for a given problem and the criteria to stop corrections if it does not create an improvement. These questions are answered in [7].

The maximum number of centrality corrections, K allowed when solving a given problem depends on the ratio of the factorization and solve efforts $r_{f/s} = E_f/E_s$. These efforts, in turn, obviously depend on the method used to solved KKT systems. Hopdm applies sparse Cholesky decomposition [5] [6]

to handle normal equations systems 10, hence the following estimates for the efforts are used

$$\begin{aligned} E_f &= \sum_{i=1}^{m} l_i^2 \\ E_s &= 2\times \sum_{i=1}^{m} l_i + 12 \times n \end{aligned} \tag{23}$$

where l_i is a number of non-zero elements in i-th row of $A\Theta A^T$. Note that the ratio $r_{f/s} = E_f/E_s$ can be determined after preprocessing the KKT system and before the optimization starts. Naturally, the greater the value of $r_{f/s}$ coefficient, the more corrections are worth trying.

The Hopdm algorithm with multiple centrality correction [7] is a nontrivial extension of the predictor corrector technique [11]. Undoubtedly, one of the reasons for the superiority of the multiple centrality corrections over the classical (second order) predictor corrector algorithm is a clever choice of targets [9] that, in our approach, do not have to be analytic centers, i.e. the perfectly centered points on the central trajectory. Instead, weighted analytic centers that stay in a wide neighborhood of the central path and can be reached much more easily are chosen.

9.3 LP_SOLVE Algorithm

The LPSOLVE uses the simplex algorithm, the implementation of the simplex kernel was mainly based on the work of W. Orchard-Hays [15]

The mixed integer branch and bound part was inspired by: section 6.4 of Paul R. Thie [18] and Dakin, R.J. [2]

9.4 Fortran library hopdm

HOPDM1 subroutine This routine calls the following subroutines of the library :

PRESOL subroutine performs an advanced pre_solve analysis of the problem.

In particular, it

- cleans the LP matrix :
  - determines (and later tightens) bounds on shadow prices,
  - eliminates dominated (and weakly dominated) variables,
  - eliminates singleton rows,
  - eliminates singleton columns (implied free variables),
  - finds identical columns and aggregates them,
  - finds hidden split free variables,
  - eliminates redundant (dominated or forcing) constraints,
  - tightens bounds on variables,
- makes the LP matrix sparser:
  - pivots out some nonzero entries of A,
- makes the LP matrix better suited for Cholesky factor.:
  - splits dense columns into shorter pieces.

PREPRO subroutine performs preprocessing for the Cholesky decomposition.

In particular, it:

- builds an adjacency structure of A*Atranspose matrix,
- finds an ordering that minimizes the number of nonzero entries in a Cholesky factor,

- reorders rows of A (and all data associated to rows, such as RANGES, RHS etc.) according to the permutation resulting from the minimum degree ordering,
- prepares data structures for sparse Cholesky decomposition (i.e. it does the symbolic factorization).

SCALEA subroutine scales the LP constraint matrix. Simple geometric scaling is repeated twice on the matrix A. RSCALE and CSCALE vectors handle the resulting row and column scaling factors, respectively.

PCPDM subroutine (Higher Order Primal-Dual Method) solves the LP problem.

This routine implements a primal-dual algorithm with multiple centrality corrections.

UNSCLA subroutine unscales the LP problem.

POSTSL subroutine performs post-solve analysis, i.e., it undoes all the problem modifications that have been done in a PRESOL routine.

Note: Generally speaking, all routines of the HOPDM library are well documented source files. To fully understand some functions, however, you may find it necessary to consult the appropriate publications (their list is almost always supplied in a source code of each routine).

10 Conclusions

This paper is based on the utilization of solvers LIPSOL[20], HOPDM[8], LP_SOLVE[17], from Scilab.

The interfaces and the proper adaptations have been developed in order to permit the solvers to be used as Scilab functions. The possibities of use are shown and their performances are compared.

The codes can be used in the resolution of multicommodity problems. Metanet (toolbox Scilab) permits an interactive resolution which involves a data introduction and graphic visualization of the results in a user-friendly way.

References

[1] A.Armacost and S. Mehrotra. Computational comparison of the network simplex method with the affine scaling method. Operations Research, 28 : 26-43, 1991.

[2] R.J., Dakin, "A Tree Search Algorithm for MILP Problems", Comput. J., 8 (1965) pp. 250-255

[3] C. Gomez M. Goursat, Metanet User's Guide and Tutorial, Manual version 1.1 for Scilab 2.4. INRIA, 1998.

[4] M. Gondran, M. Minoux, Graphes et algorithmes. Collection de la Direction des Études et Recherches d'Électricité de France. Eyrolles, 1995.

[5] J. Gondzio, Splitting dense columns of constraint matrix in interior point methods for large scale linear programming, Optimization 24: 285:297, 1992.

[6] J. Gondzio, Implementing Cholesky factorization for interior point methods of linear programming, Optimization 25 : 121-140, 1993.

[7] J. Gondzio, Multiple centrality corrections in a primal-dual method for lineal programming, Technical Report 1994.3, Department of Management Studies, University of Geneva, Geneva, Switzerland, 1994.

[8] J. Gondzio, Multiple centrality corrections in a primal-dual method for lineal programming, Computational Optimization and Application, 6, pp. 137-156, 1996.

[9] B. Jansen, C. Roos, T. Terlaky and J. Vial, Primal-dual target following algorithms for linear programmimg, Technical Mathematics and Informatics, Technical University Delf, Delf, The Netherlands, 1993. Special issue of Annals of Operation Research, K. Anstreicher and R. Freund (eds.).

[10] I.J. Lustig, R.E. Marten, and D.F. Shanno, On implementing Mehrotra's predictor corrector interior point method for linear programming, SIAM J. Optimation 2, 435-449, 1992.

[11] S. Mehrotha, On the implementation of a primal-dual interior point method, SIAM J. Optimization 2. 575-601, 1992.

[12] B.A. Murtagh, Advanced linear programming: computation and practice. McGraw-Hill International Book Co., New York, 1981.

[13] J.L. Nazareth, Computer Solutions of Linear Programs. Oxford University Press 1987.

[14] Y. Nesterov, A. Nemirovskii, Interior-point polynomial algorithms in convex programming. SIAM Studies in Applied Mathematics, 13. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1994.

[15] W. Orchard-Hays: "Advanced Linear Programming Computing Techniques", McGraw-Hill 1968

[16] H.E. Rubio Scola, Implementation of Lipsol in Scilab. Rapport Technique No 0215, INRIA, 1997.

[17] Schwab, H. Documentation for lp_solve. 1996

[18] P. R. Thie, An Introduction to Linear Programming and Game Theory, second edition, John Wiley and Sons, 1988.

[19] Y. Zhang, and D. Zhang, On polynomiality of the Mehrotha type predictor corrector interior point algorithms, Mathematical Programming 68, 303-318, 1995.

[20] Y. Zhang, User's Guide to LIPSOL, Department of Mathematics and Statistics. University of Maryland Baltimore County. USA, 1995.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399