

COIN OR Project

(Computational Infrastructure for Operations Research)

Dimitris Dimou
Alexandros Zigouris

University of Patras
mijuomij@gmail.com

April 10, 2016

Overview

CLP for Linear Programming

Installation

Overview

Usage

Executable

Installation

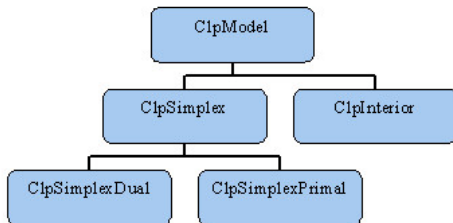
1. `svn co https://projects.coin-or.org/svn/Clp/stable/1.16`
`coin-Clp`
2. `cd coin-Clp`
3. `./configure -C`
4. `make`
5. `make test`
6. `make install`
7. `make doxydoc` (documentation)

Background

Clp is written in C++ and is released as open source code under the Eclipse Public License (EPL). It is available from the COIN-OR initiative. The code is written primarily by John J. Forrest, now retired from IBM Research. The project is currently managed by John Forrest, Julian Hall, and the rest of the Clp team. The latest stable version is 1.16.

Basic model classes

The top three levels of the hierarchy are depicted in the figure below. The first two levels (i.e. ClpModel, ClpSimplex, ClpInterior) contain all the problem data which define a model (that is, a problem instance). The third level contains most of the algorithmic aspects of CLP.



Load model

► Load from matrix

```
void ClpModel::loadProblem (const ClpMatrixBase &matrix,  
                           const double *collb,  
                           const double *colub,  
                           const double *obj,  
                           const double *rowlb,  
                           const double *rowub,  
                           const double *rowObjective = NULL)
```

► Load from MPS file

```
int ClpModel::readMps (const char *filename,  
                      bool keepNames = false,  
                      bool ignoreErrors = false)
```

► Load from GMPL file

```
int ClpModel::readGMPL (const char *filename,  
                       const char *dataName,  
                       bool keepNames = false)
```

MPS format

NAME		DOVETAIL			
ROWS					
N	obj				
L	c1				
L	c2				
L	c3				
L	c4				
COLUMNS					
	MARK0000	'MARKER '		'INTORG '	
x1	obj		3	c1	1
x1	c2		3	c3	1
x2	obj		2	c1	1
x2	c2		1	c4	1
	MARK0001	'MARKER '		'INTEND '	
RHS					
RHS	c1		9	c2	18
RHS	c3		7	c4	6
BOUNDS					
LO BND	x1		0		
LO BND	x2		0		
ENDATA					

```
#include "ClpSimplex.hpp"

int main (int argc, const char *argv[])
{
    ClpSimplex model;
    int status;
    if (argc < 2)
        status=model.readMps("dovetail.mps");
    else
        status=model.readMps(argv[1]);
    if (!status) {
        model.primal();
    }
    return 0;
}
```


Solution inspection

- ▶ `double *` `model.primalColumnSolution();`
- ▶ `double *` `model.primalRowSolution();`
- ▶ `bool` `model.isProvenOptimal();`
- ▶ `bool` `model.isProvenPrimalInfeasible();`
- ▶ `bool` `model.isProvenDualInfeasible();`
- ▶ `bool` `model.isIterationLimitReached();`

Other useful methods

Set methods

- ▶ `model.setMaximumIterations(int value);`
- ▶ `model.setMaximumSeconds(double value);`
- ▶ `model.setDualBound(double value);`
- ▶ `model.setOptimizationDirection(double value);`

Get methods

- ▶ `model.numberRows();`
- ▶ `model.numberColumns();`
- ▶ `model.objectiveValue();`
- ▶ `model.objective();`

Pivot choices

- ▶ **ClpPrimalColumnPivot**
 - ▶ ClpPrimalColumnSteepest
 - ▶ ClpPrimalColumnDantzig
- ▶ **ClpDualRowSteepest**
 - ▶ ClpDualRowDantzig
 - ▶ ClpDualRowPivot

Quick start

► Commands

- import (filename)
- primals
- duals
- max
- min
- maxIt (num)
- presolve (on/off)
- allslack
- solution (filename)
- quit

► Example :

```
clp filename -maximize -dualsimplex -solution solfile
```