Modeling with COIN-OR Tools

Leo Lopes et al

July 19, 2006

Passing an instance to a Coin Solver

Simplest example: mps or lp file $\longrightarrow$ cbc command line

Using a Traditional Modeling Language

Accessing Coin from AMPL

GAMS

A Pure C++ approach using FlopC++

Direct CBC/OSI Control

Using a Web Service

Useful Links

Getting help and reading problems in

- The command line tool is fairly spartan. cbc is designed primarily for use as a library.
  To get help on a command, type the command followed by ?

```
Enter ? for list of commands or help
Coin:import?
import : Import model from mps file
Coin:import gt2
At line 15 NAME          GT2
...
Problem GT2 has 29 rows, 188 columns and 376 elements
Model was imported from ./gt2 in -7.92823e-19 seconds
Coin:
```

Solving command and messages

- *solve*, *primalSimplex*, *dualSimplex*, ...: solves the problem using the desired method. *solve* does Branch & Bound and provides valuable information

```
Coin:solve
Cgl0004I processed model has 28 rows, 173 columns
 (173 integer) and 346 elements
...
TwoMirCuts was tried 6 times and created 128 cuts of
 which 8 were active after adding rounds of cuts
 ( 0.012001 seconds)
Result - Finished objective 21166 after 0 nodes and
 101 iterations - took 0.380024 seconds
```

Printing results

- *solution*: prints all nonzero variables to a file or stdout

```
Coin:solu £
  82 x...0309        1      9.7109072e-14
  85 x...0609        3      1.0425477e-13
...
 176 x...1114        2     -6.1756156e-16
Coin: solu outputFile
Coin:
```

Coin Support in AMPL

LP & IP cbc has support through its own plugin (driver)

- SOS, priorities, algorithms, etc.
- some features supported through suffixes, other through options.

NLP IPOPT (continuous), BONMIN (mixed-integer)

The ampl plugin (driver) system

1. ampl creates a special instance file from a model and invokes the solver as a subprocess, through a plugin (a driver in ampl terminology)
2. The plugin uses a special library to read the instance and populate the solver's data structure
3. The plugin invokes the solver
4. The plugin uses the library to reformat the solver output in a way that ampl can interpret it
5. When the plugin exits, ampl reads another file to populate its data structures

Key Point: So long as both the solver and the plugin can be found (i.e., they are on the users' path), there should be no problem

Installing the cbc or ipopt plugin (driver)

1. Get the package (substitute Ipopt or Cbc for Pkg below):

   ```
   svn co https://projects.coin-or.org/svn/Pkg/trunk pkg
   ```

2. Get the AMPL ASL library. There is a script with the distribution that makes setting up the ASL very convenient:

   ```
   cd cbc/ThirdParty/ASL
   ./get.ASL
   ```

   (for Ipopt, there are other external packages needed)

3. Run configure from the root directory of the source distribution (pkg in this case) and build.

   ```
   cd ../..
   ./configure --prefix=/usr/local
   make
   make install
   ```

Running cbc or ipopt from AMPL

Make sure the executable is on your path. Then simply choose cbc or ipopt as the solver in AMPL:

- cbc example:

```
ampl: model ablu.mod; data ablu.dat;
ampl: option solver cbc;
ampl: option cbc_options "cuts=root log=2 feas=on slog=1"
ampl: solve;
ampl: display x;
```

- to see all the possible cbc options accessible from ampl, try on the command line: `cbc -verbose 7 -? | less`
- to see all the possible ipopt options accessible from ampl, see: `http://www.coin-or.org/Ipopt/IPOPT_options.html`

Coin Support in GAMS

LP & IP cbc has support out of the box

- cbc is accessible through its old name (sbb). In GAMS: coinsbb
- GLPK also accessible via an OSI interface: coinglpk

NLP No support as of version 22 of GAMS for any Coin Nonlinear Solver

Running cbc from GAMS

1. Choose cbc as your default solver at install time

```
LP (Linear Programming) models can be solved by:
...
   3.        CoinCbc  (demo or student license)
   4.      CoinGlpk  (demo or student license)
...
Enter number for default, or hit enter
 for previous default of CPLEX: 3
```

   Make similar choices for MIP and RMIP
2. Select cbc on the command-line

```
gams transport.1 lp=coinsbb
```

3. Select cbc from within your model

```
option lp=coinsbb
```

   You can also pass options to cbc using the special variables m.integer1, m.integer2, and m.integer3

Using Coin solvers from FlopC++ via OSI

- FlopC++ supports OSI natively. There is nothing different to do
- To select different solvers, simply pass the OsiSolverInteface desired:

```
#include "flopc.hpp"
using namespace flopc;
#include <OsiCbcSolverInterface.hpp>
...
class Paper : public MP_model {
public:
    MP_set WIDTHS,PATTERNS;
...
    Paper(int numWidths) :
MP_model(new OsiCbcSolverInterface),
          WIDTHS(numWidths), PATTERNS(numWidths),
...
```

Controlling the solution process

Column Generation in FlopC++

- Problems can be resolved without being regenerated

```
 do {
ob = knapsack(numWidths, tabWidth,
    &Paper.rowPrice[demC.offset], maxWidth, pat);

    CoinPackedVector Pat;
    Pat.setFull(numWidths, pat);

    Paper->addCol(Pat,0,100,1);
    Paper->resolve();
} while(ob>1.0001);
```

Using Coin solvers directly via OSI

- Typically, build your application to a generic solver

```
class UFL {
private:
  OsiSolverInterface * si;
...
```

- Fill in the Coin and Osi data structures with A, rhs, c, etc.

```
CoinPackedMatrix * matrix =
   new CoinPackedMatrix(false,0,0);
matrix->setDimensions(0, n_cols);
for (i = 0; i < M; i++) {  //demand constraints:
  CoinPackedVector row;
  for (j = 0; j < N; j++) row.insert(xindex(i,j), 1.0);
  matrix->appendRow(row);
}
...
```

Tighten the formulation

si is a pointer to an OsiSolverInterface

```
OsiCuts cutlist;
si->setInteger(integer_vars, N);
CglGomory * gomory = new CglGomory;
gomory->setLimit(100);
gomory->generateCuts(*si, cutlist);
CglKnapsackCover * knapsack = new CglKnapsackCover;
knapsack->generateCuts(*si, cutlist);
CglSimpleRounding * rounding = new CglSimpleRounding;
rounding->generateCuts(*si, cutlist);
CglOddHole * oddhole = new CglOddHole;
oddhole->generateCuts(*si, cutlist);
CglProbing * probe = new CglProbing;
probe->generateCuts(*si, cutlist);
si->applyCuts(cutlist);
```

Solve!

si is a pointer to an OsiSolverInterface

- Instantiate the si pointer

```
si = new OsiCbcSolverInterface();
```

- solve!

```
si->branchAndBound();
```

- Get Results

```
double *sol = si->getColSolution();
```

Access a Coin Solver over a web service

Soon to be part of COIN-OR!

1. Generate an OSiL file – an XML representation of an instance, and encode solver options into an OSoL file.
2. Invoke the server remotely

```
string osil, osol, osrl;
...
string svc = "128.135.130.17:8080/os/clp/ClpSolverService.jw
OSSolverAgent* osagent = new OSSolverAgent(svc);
// this is the synchronous solver invocation method.
osrl = osagent->solve(osil, osol);
// print the result
cout << osrl  << endl;
```

Useful Links

- projects.coin-or.org/Cbc/wiki/FAQ
- www.gams.com/gamscoin
- projects.coin-or.org/FlopC++
- Ted and Matt's talk @ EURO '06
- gsbkip.chicagogsb.edu/ostalks/ostalks.html