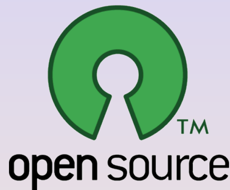
The COIN-OR Optimization Suite:

Open Source Tools for Optimization
Part 2: Developing with COIN

Ted Ralphs

LEHIGH
UNIVERSITY

COR@L
COMPUTATIONAL OPTIMIZATION RESEARCH AT LEHIGH



INFORMS Computing Society Biennial Meeting
Richmond, VA, 10 January 2015

Outline

1. COIN in Modeling Systems
2. Command-line Tools
3. Building Applications
   - Solver APIs
   - Frameworks

Outline

1. COIN in Modeling Systems
2. Command-line Tools
3. Building Applications
   - Solver APIs
   - Frameworks

Using COIN with a Modeling Language

- Commercial
  - GAMS ships with COIN solvers included,
  - MPL ships with CoinMP (wrapper around Clp and Cbc),
  - AMPL works with OSAmplClient (as well as several other projects directly),
  - AIMMS can be connected via the `AIMMSLinks` project.
- Python-based Open Source Modeling Languages and Interfaces
  - yaposib (OSI bindings)
  - CyLP (provides API-level interface)
  - PuLP/Dippy (Decomposition-based modeling)
  - Pyomo (full-featured algebraic modeling language)
- Other
  - FLOPC++ (algebraic modeling in C++)
  - CMPL (modeling language with GUI interface)
  - MathProg.jl (modeling language built in Julia)
  - GMPL (open-source AMPL clone)
  - ZMPL (stand-alone parser)
  - OpenSolver (spreadsheet plug-in)
  - Open Office
  - R (RSymphony Plug-in)
  - Matlab (OPTI)
  - Mathematica

Optimization Services (OS)

Optimization Services (OS) integrates numerous COIN-OR projects and is a good starting point for many use cases. The OS project provides:

- A set of XML based standards for representing optimization instances (`OSiL`), optimization results (`OSrL`), and optimization solver options (`OSoL`).
- A uniform API for constructing optimization problems (linear, nonlinear, discrete) and passing them to solvers.
- A command line executable `OSSolverService` for reading problem instances in several formats and calling a solver either locally or remotely.
- Utilities that convert files in AMPL nl, MPS, and LP format to OSiL.
- Client side software for creating Web Services SOAP packages with OSiL instances and contact a server for solution.
- Standards that facilitate the communication between clients and solvers using Web Services.
- Server software that works with Apache Tomcat.
- Developers: Kipp Martin, Gus Gassmann, and Jun Ma

Using AMPL with OS

To use OS to call solvers in AMPL, you specify the `OSAmplClient` as the solver.

```
model hs71.mod;
# tell AMPL that the solver is OSAmplClient
option solver OSAmplClient;

# now tell OSAmplClient to use Ipopt
option OSAmplClient_options "solver ipopt";

# now solve the problem
solve;
```

In order to call a remote solver service, set the solver `service` option to the address of the remote solver service.

```
option ipopt_options
  "service http://74.94.100.129:8080/OSServer/services/OSSolverService";
```

Outline

1 COIN in Modeling Systems

2 Command-line Tools

3 Building Applications
- Solver APIs
- Frameworks

Interactive Shells (SYMPHONY)

A number of solvers provide interactive shells and can also be invoked from the command line (SYMPHONY, Clp, Cbc, OS).

```
~/bin > ./symphony
==  Welcome to the SYMPHONY MILP Solver
==  Copyright 2000-2014 Ted Ralphs and others
==  All Rights Reserved.
==  Distributed under the Eclipse Public License 1.0
==  Version: 5.6
==  Build Date: Oct 24 2014
==  Revision Number: 2289

***** WELCOME TO SYMPHONY INTERACTIVE MIP SOLVER ******

Please type 'help'/'?' to see the main commands!

SYMPHONY:
```

To invoke, type command with no arguments in the `bin` directory (or click on icon). Note that shells are more capable when `readline` and `history` are available.

Interactive Shells (OS)

```
Please enter a command, or an option followed by an option value: ?

***************** VALID COMMANDS AND OPTIONS ********************
COMMANDS:
quit/exit -- terminate the executable
help/? -- produce this list of options
reset -- erase all previous option settings
list -- list the current option values
solve -- call the solver synchronously
send -- call the solver asynchronously
kill -- end a job on the remote server
retrieve -- get job result on the remote server
knock -- get job information on the remote server
getJobID -- get a job ID from the remote server

OPTIONS (THESE REQUIRE A VALUE):
osil -- the location of the model instance in OSiL format
mps  -- the location of the model instance in MPS format
nl -- the location of the model instance in AMPL nl format
osol -- the location of the solver option file in OSoL format
osrl -- the location of the solver result file in OSrL format
osplInput -- the name of an input file in OSpL format
osplOutput --  the name of an output file in the OSpL format
serviceLocation -- the  URL of a remote solver service
solver -- specify the solver to invoke
```

SYMPHONY: Solving on the Command Line

- MILP: Supported input format is MPS or LP

```
~/> clp -import ~/COIN/trunk/Data/Netlib/25fv47.mps.gz -solve -quit
~/> cbc -import ~/Data/miplib3/p0033.gz -solve -quit
~/> symphony -F ~/Data/miplib3/p0033.gz
~/> dip --MILP:Instance atm_5_10_1.mps --MILP:BlockFile atm_5_10_1.mps
```

- NLP and MINLP: Supported input format is .nl

```
~/> ipopt parinc.nl -AMPL
~/> bonmin bonminEx1.nl -AMPL
~/> couenne bonminEx1.nl -AMPL
```

- SYMPHONY can solve multi-objective MILPs—just specify two objectives in the input file (at the moment, this only works in LP format).

OS: Solving a Problem on the Command Line

- The OS project provides an single executable `OSSolverService` that can be used to call most COIN solvers.
- To solve a problem in MPS format

```
OSSolverService -mps parinc.mps
```

- The solver also accepts AMPL nl and OSiL formats.
- You can display the results in raw XML, but it’s better to print to a file to be parsed.

```
OSSolverService -osil parincLinear.osil -osrl result.xml
```

- You can then view in a browser using XSLT.
  - Copy the style sheets to your output directory.
  - Open in your browser

OS: Remote Solves

The OSSolverService can be invoked to make remote solve calls.

```
./OSSolverService osol remoteSolve2.osol serviceLocation
      http://74.94.100.129:8080/OSServer/services/OSSolverService
```

Note that in this case, even the instance file is stored remotely.

```
<osol xmlns="os.optimizationservices.org">
<general>

<instanceLocation locationType="http">
http://www.coin-or.org/OS/p0033.osil
</instanceLocation>
<solverToInvoke>symphony</solverToInvoke>
</general>

</osol>
```

OS: Specifying a Solver

```
OSSolverService -osil ../../data/osilFiles/p0033.osil
      -solver cbc
```

To solve a **linear program** set the solver options to:

- `clp`
- `dylp`

To solve a **mixed-integer linear program** set the solver options to:

- `cbc`
- `symphony`

To solve a **continuous nonlinear program** set the solver options to:

- `ipopt`

To solve a **mixed-integer nonlinear program** set the solver options to:

- `bonmin`
- `couenne`

OS: File formats

- What is the point of the OSiL format?
  - Provides a single interchange standard for all classes of mathematical programs.
  - Makes it easy to use existing tools for defining Web services, etc.
  - Generally, however, one would not build an OSiL file directly.
- To construct an OSiL file, there are several routes.
  - Use a modeling language—AMPL, GAMS, and MPL work with COIN-OR solvers.
  - Use FlopC++.
  - Build the instance in memory using COIN-OR utilities.
- There are also result and options languages for specifying options to a solver and getting results back.
- XML makes it easy to display the results in a standard templated format.
- This addresses one of the difficulties with using the linear solvers in COIN, which is parsing the output.

Outline

1. COIN in Modeling Systems

2. Command-line Tools

3. Building Applications
   - Solver APIs
   - Frameworks

Building Applications

- After mastering black box solvers, the next step is to try building a custom applications.
- There are two basic routes
  - Calling the library as a black box through the API.
  - Customizing the library through callbacks and customization classes.

1. COIN in Modeling Systems

2. Command-line Tools

3. Building Applications
   - Solver APIs
   - Frameworks

Building Applications: APIs

Using SYMPHONY API

```
#include "symphony.h"

int main(int argc, char **argv)
{
   sym_environment *env = sym_open_environment();

   sym_parse_command_line(env, argc, argv);

   sym_load_problem(env);

   sym_solve(env);

   sym_close_environment(env);

   return(0);
}
```

Linking to COIN Libraries: Distribution

- `bin`
- `lib`
  - `python2.*/site-packages`
  - `pkg-config`
- `share/coin`
  - `doc`
  - `Data`
- `include/coin`

Linking to COIN Libraries: Using `pkg-config`

- `pkg-config` is a utility available on most *nix systems.
- It helps automatically determine how to build against installed libraries.
- To determine the libraries that need to be linked against, the command is

```
PKG_CONFIG_PATH=/path/to/install/lib/pkgconfig pkg-config --libs cbc
```

- To determine the flags that should be given to the compiler, the command is

```
PKG_CONFIG_PATH=/path/to/install/lib/pkgconfig pkg-config --clfags cbc
```

- Note that the user doesn't need to know what any of the downstream dependencies are.
- Depending on the install location, may need to set the environment variable PKG_CONFIG_PATH.
- The .pc files are installed in

```
/path/to/install/location/lib/pkgconfig
```

Linking to COIN Libraries: pkg-config in a Makefile

- The pkg-config command can be used to vastly simplify the Makefiles used to build project that link with COIN.

```
LIBS = `PKG_CONFIG_PATH=/path/to/pc-files pkg-config --libs os`
CFLAGS = `PKG_CONFIG_PATH=/path/to/pc-files pkg-config --cflags os`

.cpp.o:
        $(CXX) \$(CFLAGS) -c -o file.cpp
$(EXE):
        $(CXX) \$(CFLAGS) -o app.exe $(OBJS) \$(LIBS)
```

- Note that the auto tools will automatically produce Makefiles that utilize pkg-config for examples.

Libtool Versioning (Shared Libraries)

- Libtools versioning allows smooth upgrading without breaking existing builds.
- The libtool version number indicates backward compatibility.
- Versions of the same library can be installed side-by-side (version number is encoded in the name).
- When a new version of a library is installed, codes built against the older library are automatically linked to the new version (if it is backward compatible).
- Based on concepts of *age*, *current*, and *revision*.

A Note About Configuration Headers

- One of the most recent enhancements to the build system is better handling of configuration header files.
- These are the files that contain settings specific to a platform or individual user's set-up.
- In all cases, the header file to include to get these settings is called `ConfigXxx.h`. From this file, the proper additional file will be included.
- For each project, the defined symbols are now divided into public and private sets, with a generated and default header for each set.
  - `config.h` (private)
  - `config_default.h` (private)
  - `config_xxx.h` (public)
  - `config_xxx_default.h` (public)
- Which header to include is controlled by whether the symbol `XXX_BUILD` is defined or not.

Finding Code Snippets and Examples

- Many projects have a directory with examples that show how to link to the library.
- The examples typically reside in the `examples/` directory of the project's source tree.
- In the near future, they will be installed as part of the binary distribution.
- If you build from source on a *nix platform, custom Makefiles are produced that allow easy linking to installed libraries.
- Visual Studio project files are also available for many examples.

Examples (Cbc)

```
~/COIN/Cbc/examples> ls
allCuts.cpp              crew.cpp        nway.cpp
barrier.cpp              driver2.cpp     parallel.cpp
CbcBranchFollow2.cpp     driver3.cpp     pool.cpp
CbcBranchFollow2.hpp     driver4.cpp     qmip2.cpp
CbcBranchLink.cpp        driver5.cpp     qmip.cpp
CbcBranchLink.hpp        driver5.cpp~    quad2.mps
CbcBranchUser.cpp        driver6.cpp     quad.mps
CbcBranchUser.hpp        driver.cpp      repeat.cpp
CbcCompareUser.cpp       fast0507b.cpp   sample1.cpp
CbcCompareUser.hpp       fast0507.cpp    sample2.cpp
cbc_driverC_sos.c        gear.cpp        sample3.cpp
CbcSolver2.cpp           hotstart.cpp    sample4.cpp
CbcSolver2.hpp           interrupt.cpp   sample5.cpp
CbcSolver3.cpp           link.cpp        simpleBAB.cpp
CbcSolver3.hpp           longthin.cpp    sos.cpp
CbcSolverLongThin.cpp    lotsize.cpp     sudoku.cpp
CbcSolverLongThin.hpp    Makefile.in     sudoku_sample.csv
ClpQuadInterface.cpp     minimum.cpp
ClpQuadInterface.hpp     modify.cpp
~/COIN/build/Cbc/examples> cd ../build/Cbc/examples
~/COIN/build/Cbc/examples> make
```

CoinBazaar and Application Templates

- CoinBazaar is a collection of examples, utilities, and light-weight applications built using COIN-OR.
- Application Templates is a project within CoinBazaar that provides templates for different kinds of projects.
- In CoinAll, it's in the `examples` directory.
- Otherwise, get it with

```
svn co
https://projects.coin-or.org/svn/CoinBazaar/projects/ApplicationTemplates/releases/1.2.2
```

- Examples
  - Branch-cut-price
  - Algorithmic differentiation
  - Adding Cgl cuts
  - ...

Using Osi

```
#include <iostream>
#include "OSIXxx.hpp"

int main(void)
{
   // Create a problem pointer.  We use the base class here.
   OsiSolverInterface *si;

   // When we instantiate the object, we need a specific derived class.
   si = new OSIXxxSolverInterface();

   // Read in an mps file.  This one's from the MIPLIB library.
   si->readMps("p0033.mps");

   // Solve the (relaxation of the) problem
   si->initialSolve();

   // Check the solution
   if ( si->isProvenOptimal() ) {
      std::cout << "Found optimal solution!" << std::endl;
      std::cout << "Objective value is " << si->getObjValue() << std::endl;

      int n = si->getNumCols();
      const double* solution = si->getColSolution();

      // We can then print the solution or could examine it.
      for( int i = 0; i < n && i < 5; ++i )
         std::cout << si->getColName(i) << " = " << solution[i] << std::endl;
      if( 5 < n )
         std::cout << "..." << std::endl;
   } else {
      std::cout << "Didn't find optimal solution." << std::endl;
      // Could then check other status functions.
   }
}
```

Calling a Solver with OS

Step 1: Construct an instance in a solver-independent format using the OS API.

Step 2: Create a solver object

```
CoinSolver *solver = new CoinSolver();
solver->sSolverName = "clp";
```

Step 3: Feed the solver object the instance created in Step 1.

```
solver->osinstance = osinstance;
```

Step 4: Build solver-specific model instance

```
solver->buildSolverInstance();
```

Step 5: Solve the problem.

```
solver->solve();
```

Building an OS Instance

The `OSInstance` class provides an API for constructing models and getting those models into solvers.

- `set()` and `add()` methods for creating models.
- `get()` methods for getting information about a problem.
- `calculate()` methods for finding gradient and Hessians using algorithmic differentiation.

Building an OS Instance (cont.)

- Create an `OSInstance` object.

```
OSInstance *osinstance =  new OSInstance();
```

- Put some variables in

```
osinstance->setVariableNumber( 2);
osinstance->addVariable(0, "x0", 0, OSDBL_MAX, 'C', OSNAN, "");
osinstance->addVariable(1, "x1", 0, OSDBL_MAX, 'C', OSNAN, "");
```

- There are methods for constructing
  - the objective function
  - constraints with all linear terms
  - quadratic constraints
  - constraints with general nonlinear terms

Building Linear Models

- `CoinUtils` has a number of utilities for constructing instances.
  - `PackedMatrix` and `PackedVector` classes.
  - `CoinBuild`
  - `CoinModel`
- `Osi` provides an interface for building models and getting them into solvers for linear probes.

In Class Exercise: Build an Example!

1 COIN in Modeling Systems

2 Command-line Tools

3 Building Applications
- Solver APIs
- Frameworks

Customization through Callbacks and Inheritance

- A number of the solvers can be customized with callbacks for adding such things as
  - Valid inequalities
  - Heuristics
  - Branching
- These include Clp, Cbc, SYMPHONY, Bcp, DIP, and CHiPPS.
- In Dippy, callbacks can be written in Python, providing convenient customization options.
- Most other frameworks require coding in C/C++.
- On the TODO list is to enable Python callbacks in more projects.

SYMPHONY (with M. Guzelsoy and A. Mahajan)

Using SYMPHONY

- C Library API
- OSI C++ interface
- Interactive shell
- AMPL/GMPL, GAMS, FLOPC++
- Framework for customization

Advanced Features

- Shared and distributed memory parallel MIP
- Biobjective MIP
- Warm starting for MIP
- Sensitivity analysis for MIP

SYMPHONY Applications

- TSP/VRP
- Set Partitioning Problem
- Mixed Postman Problem

SYMPHONY Callbacks

```
int user_find_cuts(void *user, int varnum, int iter_num, int level,
                   int index, double objval, int *indices, double *values,
                   double ub, double etol, int *num_cuts, int *alloc_cuts,
                   cut_data ***cuts)
{
   user_problem *prob = (user_problem *) user;
   double edge_val[200][200]; /* Matrix of edge values */
   int i, j, k, cutind[3];
   double cutval[3];
   int cutnum = 0;
   memset((char *)edge_val, 0, 200*200*ISIZE);

   for (i = 0; i < varnum; i++) {
      edge_val[prob->match1[indices[i]]][prob->match2[indices[i]]] = values[i];
   }

   for (i = 0; i < prob->numnodes; i++){
      for (j = i+1; j < prob->numnodes; j++){
         for (k = j+1; k < prob->numnodes; k++) {
            if (edge_val[i][j]+edge_val[j][k]+edge_val[i][k] > 1.0 + etol) {
               /* Found violated triangle cut */
               /* Form the cut as a sparse vector */
               cutind[0] = prob->index[i][j];
               cutind[1] = prob->index[j][k];
               cutind[2] = prob->index[i][k];
               cutval[0] = cutval[1] = cutval[2] = 1.0;
               cg_add_explicit_cut(3, cutind, cutval, 1.0, 0, 'L',
                                   TRUE, num_cuts, alloc_cuts, cuts);
               cutnum++;

            }
         }
      }
   }
}
```

DIP Framework: Motivation

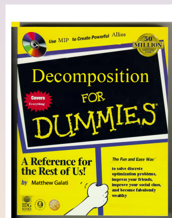
DIP Framework

DIP is a software framework and stand-alone solver for implementation and use of a variety of decomposition-based algorithms.

- Decomposition-based algorithms have traditionally been extremely difficult to implement and compare.
- DIP abstracts the common, generic elements of these methods.
  - Key: API is in terms of the compact formulation.
  - The framework takes care of reformulation and implementation.
  - DIP is now a *fully generic* decomposition-based parallel MILP solver.

Methods

- Column generation (Dantzig-Wolfe)
- Cutting plane method
- Lagrangian relaxation (not complete)
- Hybrid methods



⇐ *Joke*!

DIP Framework (with Matt Galati)

- The DIP framework, written in C++, is accessed through two user interfaces:
  - Applications Interface: `DecompApp`
  - Algorithms Interface: `DecompAlgo`
- DIP provides the bounding method for branch and bound.
- ALPS (Abstract Library for Parallel Search) provides the framework for parallel tree search.
  - `AlpsDecompModel : public AlpsModel`
    - a wrapper class that calls (data access) methods from `DecompApp`
  - `AlpsDecompTreeNode : public AlpsTreeNode`
    - a wrapper class that calls (algorithmic) methods from `DecompAlgo`

```
~/COIN/build/Dip/examples/GAP> sh get_pisinger.sh
~/COIN/build/Dip/examples/GAP> sh data_extract.sh
~/COIN/build/Dip/examples/GAP> make
~/COIN/build/Dip/examples/GAP> ./decomp_gap --param gap.parm
```

CHiPPS (with Yan Xu)

- CHiPPS stands for COIN-OR High Performance Parallel Search.
- CHiPPS is a set of C++ class libraries for implementing tree search algorithms for both sequential and parallel environments.

CHiPPS Components (Current)

ALPS (Abstract Library for Parallel Search)
- is the search-handling layer (parallel and sequential).
- provides various search strategies based on node priorities.

BiCePS (Branch, Constrain, and Price Software)
- is the data-handling layer for relaxation-based optimization.
- adds notion of variables and constraints.
- assumes iterative bounding process.

BLIS (BiCePS Linear Integer Solver)
- is a concretization of BiCePS.
- specific to models with linear constraints and objective function.

ALPS: Design Goals

- Intuitive object-oriented class structure.
  - `AlpsModel`
  - `AlpsTreeNode`
  - `AlpsNodeDesc`
  - `AlpsSolution`
  - `AlpsParameterSet`
- Minimal algorithmic assumptions in the base class.
  - Support for a wide range of problem classes and algorithms.
  - Support for constraint programming.
- Easy for user to develop a custom solver.
- Design for *parallel scalability*, but operate effective in a sequential environment.
- Explicit support for *memory compression* techniques (packing/differencing) important for implementing optimization algorithms.

ALPS: Overview of Features

- The design is based on a very general concept of *knowledge*.
- Knowledge is shared asynchronously through *pools* and *brokers*.
- Management overhead is reduced with the *master-hub-worker* paradigm.
- Overhead is decreased using dynamic task granularity.
- Two static load balancing techniques are used.
- Three dynamic load balancing techniques are employed.
- Uses asynchronous messaging to the highest extent possible.
- A scheduler on each process manages tasks like
  - node processing,
  - load balancing,
  - update search states, and
  - termination checking, etc.

BiCePS: Support for Relaxation-based Optimization

- Adds notion of *modeling objects* (variables and constraints).
- Models are built from sets of such objects.
- Bounding is an iterative process that produces new objects.
- A differencing scheme is used to store the difference between the descriptions of a child node and its parent.

```
struct BcpsObjectListMod
{
    int  numRemove;
    int* posRemove;
    int  numAdd;
    BcpsObject **objects;
    BcpsFieldListMod<double> lbHard;
    BcpsFieldListMod<double> ubHard;
    BcpsFieldListMod<double> lbSoft;
    BcpsFieldListMod<double> ubSoft;
};
```

```
template<class T>
struct BcpsFieldListMod
{
    bool relative;
    int  numModify;
    int  *posModify;
    T    *entries;
};
```

BLIS: A Generic Distributed Solver for MILP

MILP

$$\begin{aligned} & min && c^T x && (1) \\ & s.t. && Ax \le b && (2) \\ & && x_i \in \mathbb{Z} \quad \forall\, i \in I && (3) \end{aligned}$$

where $(A, b) \in \mathbb{R}^{m \times (n+1)}$, $c \in \mathbb{R}^n$.

Basic Algorithmic Components

- Bounding method.
- Branching scheme.
- Object generators.
- Heuristics.

In Class Exercise: Build an Application!

End of Part 2!

Questions?