Computational Integer Programming

Lecture 3: Software

Dr. Ted Ralphs

LEHIGH
UNIVERSITY

open source

COR@L
COMPUTATIONAL OPTIMIZATION
RESEARCH AT LEHIGH

Introduction to Software (Solvers)

- There is a wealth of software available for modeling, formulation, and solution of MILPs.
- Commercial solvers
  - IBM CPLEX
  - FICO XPress
  - Gurobi
- Open Source and Free for Academic Use
  - COIN-OR Optimization Suite (Cbc, SYMPHONY, Dip)
  - SCIP
  - lp-solve
  - GLPK
  - MICLP
  - Constraint programming systems

Introduction to Software (Modeling)

- There are two main categories of modeling software.
  - Algebraic Modeling Languages (AMLs)
  - Constraint Programming Systems (CPSs)
- According to our description of the modeling process, AMLs should probably be called "formulation languages".
- AMLs assume the problem will be formulated as a mathematical optimization problem.
- Although AMLs make the formulation process more convenient, the user must provide an initial "formulation" and decide on an appropriate solver.
- Solvers do some internal reformulation, but this is limited.
- Constraint programming systems use a much higher level description of the model itself.
- Reformulation is done internally before automatically passing the problem to the most appropriate of a number of solvers.

Algebraic Modeling Languages

- A key concept is the separation of "model" (formulation, really) from "data."
- Generally speaking, we follow a four-step process in modeling with AMLs.
  - Develop an "abstract model" (more like a formulation).
  - Populate the formulation with data.
  - Solve the Formulation.
  - Analyze the results.
- These four steps generally involve different pieces of software working in concert.
- For mathematical optimization problems, the modeling is often done with an *algebraic modeling system*.
- Data can be obtained from a wide range of sources, including spreadsheets.
- Solution of the model is usually relegated to specialized software, depending on the type of model.

Introduction to Software (AMLs)

- Commercial AMLs.
  - AMPL
  - GAMS
  - MPL
- 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 Open Source and Free for Academic Use AMLs
  - FLOPC++ (algebraic modeling in C++)
  - CMPL (modeling language with GUI interface)
  - MathProg.jl (modeling language built in Julia)
  - GMPL (open-source AMPL clone)
  - ZIMPL (stand-alone parser)

Introduction to Software (Other Interfaces)

- SolverStudio (spreadsheet plug-in)
- Open Office
- R (RSymphony Plug-in)
- Matlab (OPTI)
- Mathematica
- Optimization Services
- Sage

How AMLs Interface

- Although not required, it's useful to know something about how modeling languages interface with solvers.
- In many cases, modeling languages interface with solvers by writing out an intermediate file that the solver then reads in.
- It is also possible to generate these intermediate files directly from a custom-developed code.
- Common file formats
  - MPS format: The original standard developed by IBM in the days of Fortran, not easily human-readable and only supports (integer) linear modeling.
  - LP format: Developed by CPLEX as a human-readable alternative to MPS.
  - .nl format: AMPL's intermediate format that also supports non-linear modeling.
  - OSIL: an open, XML-based format used by the Optimization Services framework of COIN-OR.
  - Python C Extension: Several projects interface through a Python extension that can be easily

Software We'll Install

- COIN-OR Optimization Suite
- SCIP Optimization Suite
- AMPL Trial Version
- Python
  - PuLP
  - Pyomo
  - DiPPy (maybe)
  - GiMPy/GrUMPy

Brief Introduction to COIN-OR

- The COIN-OR Foundation
  - A non-profit foundation promoting the development and use of interoperable, open-source software for operations research.
  - A consortium of researchers in both industry and academia dedicated to improving the state of computational research in OR.
  - A venue for developing and maintaining standards.
  - A forum for discussion and interaction between practitioners and researchers.
- The COIN-OR Repository
  - A collection of interoperable software tools for building optimization codes, as well as a few stand alone packages.
  - A venue for peer review of OR software tools.
  - A development platform for open source projects, including a wide range of project management tools.

See http://www.coin-or.org for more information.

What You Can Do With COIN-OR: Low-level Tools

- We currently have 50+ projects and more are being added all the time.
- Most projects are now licensed under the EPL (very permissive).
- COIN-OR has solvers for most common optimization problem classes.
  - Linear programming
  - Nonlinear programming
  - Mixed integer linear programming
  - Mixed integer nonlinear programming (convex and nonconvex)
  - Stochastic linear programming
  - Semidefinite programming
  - Graph problems
  - Combinatorial problems (VRP, TSP, SPP, etc.)
- COIN-OR has various utilities for reading/building/manipulating/preprocessing optimization models and getting them into solvers.
- COIN-OR has overarching frameworks that support implementation of broad algorithm classes.
  - Parallel search
  - Branch and cut (and price)
  - Decomposition-based algorithms

What You Can Do With COIN-OR: High-level Tools

One of the most exciting developments of recent years is the number of is the wide range of high-level tools available to access COIN-OR solvers.

- Python-based modeling languages
- Spreadsheet modeling (!)
- Commercial modeling languages
- Mathematica
- Matlab
- R
- Sage
- Julia
- Optimization Services
- ...

COIN-OR Optimization Suite: Modular Structure

- One of the hallmarks of good open source tools is *modularity*.
- The suite is made up of building blocks with well-defined interfaces that allow construction of higher level tools.
- There have been 75 authors over time and most have never coordinated directly with each other!
- This is the open source model of development.

Basic Building Blocks: CoinUtils

The CoinUtils project contains a wide range of low-level utilities used in almost every project in suite.

- Factorization
- File parsing
- Sparse matrix and array storage
- Presolve
- Memory management
- Model building
- Parameter parsing
- Timing
- Basic data structures

Basic Building Blocks: Open Solver Interface

Uniform API for a variety of solvers:

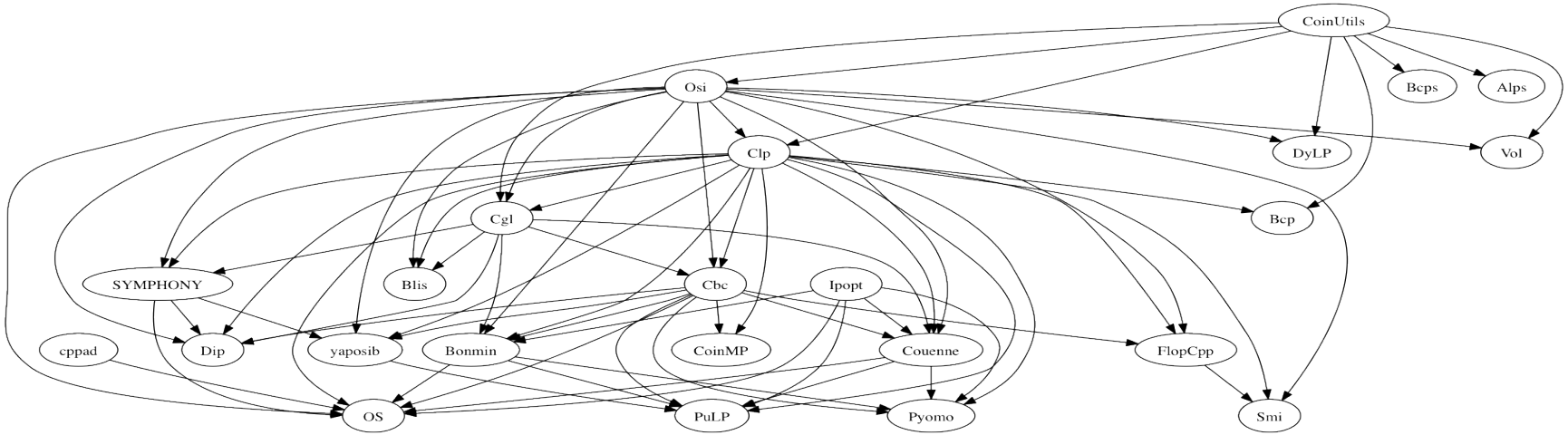
- CBC
- CLP
- CPLEX
- DyLP
- FortMP
- XPRESS-MP
- GLPK
- Mosek
- OSL
- Soplex
- SYMPHONY
- Volume Algorithm

- Read input from MPS or CPLEX LP files or construct instances using COIN-OR data structures.
- Manipulate instances and output to MPS or LP file.
- Set solver parameters.
- Calls LP solver for LP or MIP LP relaxation.
- Manages interaction with dynamic cut and column generators.
- Calls MIP solver.
- Returns solution and status information.

Building Blocks: Cut Generator Library

- A collection of cutting-plane generators and management utilities.
- Interacts with OSI to inspect problem instance and solution information and get violated cuts.
- Cuts include:
  - Combinatorial cuts: AllDifferent, Clique, KnapsackCover, OddHole
  - Flow cover cuts
  - Lift-and-project cuts
  - Mixed integer rounding cuts
  - General strengthening: DuplicateRows, Preprocessing, Probing, SimpleRounding

Optimization Suite Dependency Graph



Installing the COIN-OR Optimization Suite

- Many of the tools mentioned interoperate by using the configuration and build utilities provided by the `BuildTools` project.
- The `BuildTools` project provides build infrastructure for
  - MS Windows (CYGWIN, MINGW, and Visual Studio)
  - Linux
  - Mac OS X (clang, gcc)
- The `BuildTools` provides `autoconf` macros and scripts to allow the modular use of code across multiple projects.
- If you work with multiple COIN projects, you may end up maintaining many (possibly incompatible) copies of COIN libraries and binaries.
- The easiest way to use multiple COIN projects is simply to download and install the latest version of the suite (`1.8` due out imminently).
- The `TestTools` project is the focal point for testing of COIN code.

Getting the Binary Distribution

- The CoinBinary project is a long-term effort to provide pre-built binaries and installers for popular platforms.
- You can download some binaries here (may be out of date):

  http://www.coin-or.org/download/binary/OptimizationSuite
  http://ampl.com/products/solvers/open-source/

- Installers
  - For Windows, there is an installer available at the URL above for installing libraries compatible with Visual Studio (you will need to install the free Intel compiler redistributable libraries).
  - For OS X, there are Homebrew recipes for some projects (we are working on adding more).
  - For Linux, there are now Debian and Fedora packages for most projects in the suite and we are investigating the possiblity of providing Linuxbrew packages
- Other ways of obtaining COIN include downloading it through a number of modeling language front-ends (more on this later).

Getting the Source

- Why download and build COIN yourself?
  - There are many options for building COIN codes and the distributed binaries are built with just one set of options.
  - We cannot distribute binaries linked to libraries licensed under the GPL, so you must build yourself if you want GMPL, command completion, command history, Haskell libraries, etc.
  - Other advanced options that require specific hardware/software my also not be supported in distributed binaries (parallel builds, MPI)
  - Once you understand how to get and build source, it is *much* faster to get bug fixes.
- You can download source tarballs and zip archives here:

  `http://www.coin-or.org/download/source/OptimizationSuite`
- The recommended way to get source is to use `subversion`, although `git` is also an option.
- With subversion, it is easy to stay up-to-date with the latest sources and to get bug fixes.

  `http://www.coin-or.org/svn/CoinBinary/OptimizationSuite`

What Version to Get?

- About version numbers
  - COIN numbers versions by a standard semantic versioning scheme: each version has a *major*, *minor*, and *patch/release* number.
  - All version within a *major.minor* series are compatible.
  - All versions within a *major* series are backwards compatible.
- Organization of the repositories
  - At the top level, all repositories have the following directory structure.

```
html/
conf/
branches/
trunk
stable/
releases/
```

  - Trunk is where development takes place (bleeding edge).
  - Stable versions have two digits and are continuously patched with fixes and updates.
  - Release versions have three digits and are fixed forever.
- If you are using subversion to get code, you want the latest stable version.
- If you are downloading a tarball, you want the latest release.

Build Tools

- The primary build system is based on the GNU auto tools (there is a CMake harness being developed).
  - Build scripts work on any platform
  - Externals can be used to get complete sources (including dependencies).
  - Projects are only loosely coupled and can be installed individually.
  - Scripts available for upgrading to latest releases.
  - Smooth upgrade path.
- Features
  - Libtool library versioning.
  - Support for pkg-config.
  - Build against installed binaries.
  - Wrapper libraries for third party open source projects.

Source Tree Organization

- The source tree for project Xxx looks something like:

```
Xxx/
doxydoc/
INSTALL
Dependencies
configure
Makefile.am
...
```

- The files in the root directory are for doing monolithic builds, including dependencies (listed in the `Dependencies` file).
- Source code for dependencies is pulled in using the svn externals mechanism.
- If you only want to build the project itself and lnk against installed binaries of other projects, you only need the `Xxx/` subdirectory.

Source Tree Organization (Project Subdirectory)

- The source tree for project Xxx looks something like:

```
src/
examples/
MSVisualStudio/
test/
AUTHORS
README
LICENSE
INSTALL
configure
Makefile.am
...
```

- The files in the subdirectory are for building the project itself, with no dependencies.
- The exception is the `MSVisualStudio/` directory, which contains solution files that include dependencies.

Preparing to Build on Windows

- The easiest way to build on Windows is with the GNU autotools.
- First step is to install either Msys2 or CYGWIN.
  - For MSys2, download and unzip MSys2 base:
    `http://kent.dl.sourceforge.net/project/msys2/Base/x86_64/msys2-base-x86_64-20150512.tar.xz`
  - Add `msys64\usr\bin`, `msys64\mingw32\bin`, and `msys64\mingw64\bin` to your Windows path.
  - At a Windows command prompt, do
    ```
    bash
    pacman -S make wget tar patch dos2unix diffutils svn
    ```
- For CYGWIN, download the CYGWIN installer and run it.
  - It is a bit more complicated because you have to choose your packages.
  - You need at least gcc, g++, and gfortran, and other optional packages.
  - It's helpful to install the X server (xorg) in order to have graphical interfaces, but this is not necessary.
  - Add `C:\cygwin\bin` to your `PATH`.

Preparing to Build on OS X

- The latest versions of OS X come with the clang compiler but no Fortran compiler.
- The easiest way to remedy this is to install Homebrew (see `brew.sh`) and then `brew install gcc`.
- It will also be helpful to `brew install wget`.
- Notes:
  - Since clang uses the GNU standard library, gfortran is compatible and this is what will be used to build Fortran code when required.
  - Clang will be used to build by default. If you want to use, e.g., the `gcc` compiler provided by Homebrew, you need to specify that with `CC=gcc-5 CXX=g++-5`.

Building from Source (All Platforms)

- First, open a terminal (in Windows, run `cmd` and type `bash`).
- To build the `OptimizationSuite`, first get the source by either

```
wget \
http://www.coin-or.org/download/source/CoinBinary/OptimizationSuite/\
CoinBinary-OptimizationSuite-1.8.0.zip
unzip CoinBinary-OptimizationSuite-1.8.0.zip
```

  or by

```
svn co \
http://projects.coin-or.org/svn/CoinBinary/OptimizationSuite/stable/1.8 \
OptimizationSuite-1.8
```

- Then do

```
cd OptimizationSuite-1.8
./get.allThirdParty
mkdir build
cd build
../configure COIN_SKIP_PROJECTS="FlopCpp" --enable-gnu-packages \
--prefix=/path/to/install/dir
make -j 2
make test
make install
```

After Building

- Note that in order to use the installed binaries and libraries, you will need to add the directory `OptimizationSuite-1.8/build/bin` to your executable `PATH` (or install in a system directory using `--prefix`, see below).
- If you move the installed libraries or link to them from a non-COIN binary, you need to add `OptimizationSuite-1.8/build/lib` to your `LD_LIBRARY_PATH` (`DYLD_LIBRARY_PATH` on OS X).
- Note that after building, the examples will be installed with Makefiles in project subdirectories.

Building on Windows with Visual Studio Compiler

- To build with the Visual Studio compiler
  - If you don't already have the IDE, you can download the Microsoft SDK, which includes the compilers.
  - To build any of the non-linear solvers, you will need a compatible Fortran compiler, such as the one from Intel.
  - Run `vcvarsall.bat` to set the proper environment variables (and `ifortvars.bat` for Intel compiler).
  - At a Windows command propt, do

```
bash
cd COIN-1.8
./configure --enable-msvc
make
make install
```

  - To build Python extensions, you should use the Visual Studio compiler and build with `--enable-msvc=MD`.
- To build with the GNU compilers, just open `bash` and follow the instructions for Linux/OS X.

Building on Windows (Visual Studio IDE)

- To build through the Visual Studio IDE, MSVC++ project files are provided.
  - Solution files for v10 are provided, but upgrading to other versions should work.
  - Important: Common settings are saved using property sheets!!
  - Change the settings on the property sheets, not in the individual projects and configurations!!!!
  - It is incredibly easy to slip up on this and the repercussions are always annoyingly difficult to deal with.

ThirdParty Projects

- There are a number of open-source projects that COIN projects can link to, but whose source we do not distribute.
- We provide convenient scripts for downloading these projects (shell scripts named `./get.Xxx`) and a build harness for build them.
- We also produce libraries and pkg-config files.
  - AMPL Solver Library (required to use solvers with AMPL)
  - Blas (improves performance—usually available natively on Linux/OS X)
  - Lapack (same as Blas)
  - Glpk
  - Metis
  - MUMPS (required for Ipopt to build completely open source)
  - Soplex
  - SCIP
  - HSL (an alternative to MUMPS that is not open source)
  - FilterSQP

Parallel Builds

- SYMPHONY, DIP, CHiPPS, and Cbc all include the ability to solve in parallel.
  - CHiPPS uses MPI and is targeted at massive parallelism (it would be possible to develop a hybrid algorithm, however).
  - SYMPHONY and Cbc both have shared memory threaded parallelism.
  - DIP's parallel model is still being implemented but is a hybrid distributed/shared approach.
- To enable shared memory for Cbc, option is `--enable-cbc-parallel`.
- For SYMPHONY, it's `--enable-openmp` (now the default).
- For CHiPPS, specify the location of MIP with `--with-mpi-incdir` and `--with-mpi-lib`:

```
configure --enable-static
          --disable-shared
          --with-mpi-incdir=/usr/include/mpich2
          --with-mpi-lib="-L/usr/lib  -lmpich"
          MPICC=mpicc
          MPICXX=mpic++
```

Other Configure-time Options

- There are many configure options for customizing the builds, which is the advantage of learning to build yourself.
  - Over-riding variables: `CC`, `CXX`, `F77`, `CXX_ADDFLAGS`
  - `--prefix`
  - `--enable-debug`
  - `--enable-gnu-packages`
  - `-C`
- `configure --help` lists many of the options, but beware that configure is recursive and the individual project also have their own options.
  - `SYMPHONY/configure --help` will list the options for SYMPHONY.
  - These options can be given to the root `configure`—they will be passed on automatically.

Building Individual Packages from Source

- Assuming some libraries are already installed in `/some/dir`

```
svn co http://projects.coin-or.org/svn/Cbc/stable/2.8/Cbc Cbc-2.8
cd Cbc-2.8
mkdir build
cd build
../configure --enable-gnu-packages -C --with-coin-instdir=/some/dir
make -j 2
make test
make install
```

- Note that this checks out Cbc without externals and links against installed libraries.
- "Old style" builds will still work with all dependencies checked out using SVN externals.

Working With Git

- You can now get most of the COIN projects that are part of the Optimization Suite from github.
  `git clone https://github.com/coin-or/Xxx`
- Stables are branches and releases are tags.
  `git clone --branch=stable/X.X`
  `git clone --branch=releases/X.X.X`
- To build from source, there is a script that fetches dependent projects and build automatically.

```
git clone --branch=stable/1.8 https://github.com/coin-or-tools/BuildTools/
BuildTools/get.dependencies fetch
BuildTools/get.dependencies build --quiet --test
```

Documentation

- Documentation on using the full optimization suite

  http://projects.coin-or.org/CoinHelp
  http://projects.coin-or.org/CoinEasy

- User's manuals and documentation for individual projects

  http://projects.coin-or.org/ProjName
  http://www.coin-or.org/ProjName

- Source code documentation

  http://www.coin-or.org/Doxygen

Support

- Support is available primarily through mailing lists and bug reports.

  `http://list.coin-or.org/mailman/listinfo/ProjName`
  `http://projects.coin-or.org/ProjName`

- Keep in mind that the appropriate place to submit your question or bug report may be different from the project you are actually using.
- Make sure to report all information required to reproduce the bug (platform, version number, arguments, parameters, input files, etc.)
- Also, please keep in mind that support is an all-volunteer effort.
- In the near future, we will be moving away from mailing lists and towards support forums.

Introduction to the SCIP Optimization Suite

- SCIP is free-for-academic-use software that provides many of the same capabilities as COIN-OR.
- It consists of the following packages.
  - SCIP
  - SoPlex (LP solver)
  - ZIMPL (stand-alone modeling language)
  - UG (parallel optimization framework)
  - GCG (decomposition-based solver)
- The easiest way to get SCIP is to download the binaries here:

`http://scip.zib.de/#download`

- You can also download source and build with `make` on the command line.

Exercise: Download and Install COIN and SCIP

```
wget https://github.com/coin-or/yaposib/blob/master/examples/p0033.mps
symphony -F p0033.mps
cbc p0033.mps
scip -f p0033.mps
```