The COIN-OR Optimization Suite:

Open Source Tools for Optimization
Part 1: Introduction and Development

Ted Ralphs

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

Intro and Caveats

- First, thank you for being here!!
- I'll touch on a lot of things and will drill down when there's interest.
- I won't always know what I'm talking about—pitch in if you know more than me!
- I'm going to talk about the work of lots of different people and will inevitably miss some attributions.
- I'll try to focus on stuff that's not so easy to find/understand.
- There are a LOT of moving parts to all of this, so things will inevitably go wrong: please be patient!
- If you like what you see, consider making a donation to the COIN-OR Foundation.
- Please ask questions and/or make corrections!

Let's Go!

Outline

Outline

Brief History of COIN-OR

- The Common Optimization Interface for Operations Research Initiative was an initiative launched by IBM at ISMP in 2000.
- IBM seeded an open source repository with four initial projects and created a Web site.
- The goal was to develop the project and then hand it over to the community.
- The project grew to be self-sustaining and was spun off as a nonprofit educational foundation in the U.S. a decade ago.
- The name was also changed to the Computational Infrastructure for Operations Research to reflect a broader mission.

What is COIN-OR Today?

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.

The COIN Boards

The COIN-OR Foundation is governed by two boards.

Strategic Leadership Board

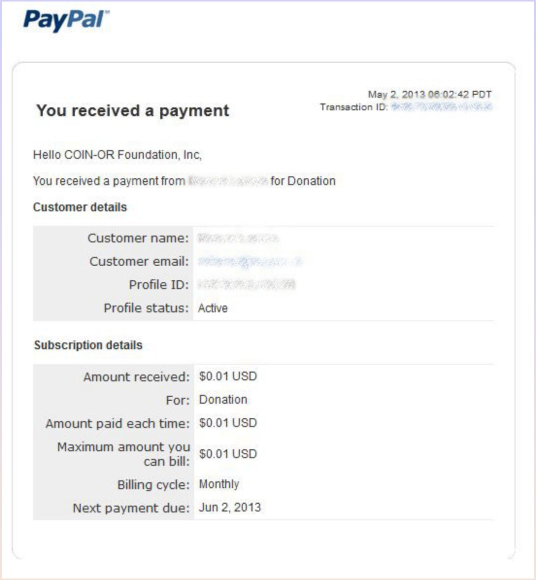
- Kevin Furman
- Lou Hafer (Secretary)
- Alan King (Treasurer)
- Andrew Mason
- Ted Ralphs (TLC Rep)
- Daniel Reich
- Matt Saltzman (President)

Technical Leadership Council

- Miles Lubin
- Ted Ralphs (Chair)
- Haroldo Santos
- John Siirola
- Mike Steglich
- Stefan Vigerske

- The SLB sets the overall strategic direction and manages the business operations: budgeting, fund-raising, legal, etc.
- The TLC focuses on technical issues: build system, versioning system, bug reporting, interoperability, etc.

How is COIN Supported?



PayPal

May 2, 2013 08:02:42 PDT
Transaction ID: [redacted]

You received a payment

Hello COIN-OR Foundation, Inc,

You received a payment from [redacted] for Donation

Customer details

Customer name:	[redacted]
Customer email:	[redacted]
Profile ID:	[redacted]
Profile status:	Active

Subscription details

Amount received:	$0.01 USD
For:	Donation
Amount paid each time:	$0.01 USD
Maximum amount you can bill:	$0.01 USD
Billing cycle:	Monthly
Next payment due:	Jun 2, 2013

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
- ...

COIN-OR isn't just for breakfast anymore!

Current Priorities

- Development efforts have been moving up the stack.
- Core tools are still evolving but emphasis has shifted to maintenance, documentation, improvements to usability, development of the ecosystem.

Current priorities

- We're now on Github and we have git hosting
- Packages on Fedora and Debian
- Homebrew installation on OS X
- Installers on Windows
- Re-launching Web site with many new features
    - Forums
    - Social integration, single sign-on (OpenID)
    - Individual project Web sites
- Modeling tools
- Python support
- And more...

Outline

COIN-OR Projects Overview: Linear Optimization

- Clp: COIN LP Solver

  Project Manager: Julian Hall
- DyLP: An implementation of the dynamic simplex method

  Project Manager: Lou Hafer
- Cbc: COIN Branch and Cut

  Project Manager: Ted Ralphs
- SYMPHONY: a flexible integer programming package that supports shared and distributed memory parallel processing, biobjective optimization, warm starting, sensitivity analysis, application development, etc.

  Project Manager: Ted Ralphs
- BLIS: Parallel IP solver built to test the scalability of the CHiPPS framework.

  Project Manager: Ted Ralphs
- Cgl: A library of cut generators

  Project Manager: Robin Lougee

COIN-OR Projects Overview: Nonlinear Optimization

- Ipopt: Interior Point OPTimizer for nonlinear optimization problems.

  Project Manager: Andreas Wächter
- DFO: An algorithm for derivative free optimization.

  Project Manager: Katya Scheinberg
- CSDP: A solver for semi-definite programs

  Project Manager: Brian Borchers
- OBOE: Oracle based optimization engine

  Project Manager: Nidhi Sawhney
- FilterSD: Package for solving linearly constrained non-linear optimization problems.

  Project Manager: Frank Curtis
- OptiML: Optimization for Machine learning, interior point, active set method and parametric solvers.

  Project Manager: Katya Scheinberg
- qpOASES: QP solver using the active online set strategy.

  Project Manager: Joachim Ferreau

COIN-OR Projects Overview: Mixed Integer Nonlinear Optimization

- Bonmin: Basic Open-source Nonlinear Mixed INteger programming is for (convex) nonlinear integer programming.

  Project Manager: Pierre Bonami

- Couenne: Solver for nonconvex nonlinear integer programming problems.

  Project Manager: Pietro Belotti

- LaGO: Lagrangian Global Optimizer, for the global optimization of nonconvex mixed-integer nonlinear programs.

  Project Manager: Stefan Vigerske

COIN-OR Projects Overview: Modeling

- FLOPC++: An open-source modeling system.
  Project Manager: Tim Hultberg
- Pyomo: A repository of python-based modeling tools.
  Project Manager: Bill Hart
- PuLP: Another python-based modeling language.
  Project Manager: Stu Mitchell
- DipPy: A python-based modeling language for decomposition-based solvers.
  Project Manager: Mike O'Sullivan
- CMPL: An algebraic modeling language
  Project Manager: Mike Stieglich
- SMI: Stochastic Modeling Interface, for optimization under uncertainty.
  Project Manager: Alan King
- yaposib: Yet Another Python OSI Binding.
  Project Manager: Ted Ralphs
- CyLP: Python interface to Cbc and Clp.
  Project Manager: Mehdi Towhidi

COIN-OR Projects Overview: Interfaces and Solver Links

- Osi: Open solver interface is a generic API for linear and mixed integer linear programs.

  Project Manager: Matthew Saltzman
- GAMSlinks: Allows you to use the GAMS algebraic modeling language and call COIN-OR solvers.

  Project Manager: Stefan Vigerske
- AIMMSlinks: Allows you to use the AIMMS modeling system and call COIN-OR solvers.

  Project Manager: Marcel Hunting
- MSFlinks: Allows you to call COIN-OR solvers through Microsoft Solver Foundation.

  Project Manager: Lou Hafer
- CoinMP: A callable library that wraps around CLP and CBC, providing an API similar to CPLEX, XPRESS, Gurobi, etc.

  Project Manager: Bjarni Kristjansson
- Optimization Services: A framework defining data interchange formats and providing tools for calling solvers locally and remotely through Web services.

  Project Managers: Jun Ma, Gus Gassmann, and Kipp Martin

COIN-OR Projects Overview: Frameworks

- Bcp: A generic framework for implementing branch, cut, and price algorithms.

  Project Manager: Laci Ladanyi

- CHiPPS: A framework for developing parallel tree search algorithms.

  Project Manager: Ted Ralphs

- DIP: A framework for implementing decomposition-based algorithms for integer programming, including Dantzig-Wolfe, Lagrangian relaxation, cutting plane, and combinations.

  Project Manager: Ted Ralphs

COIN-OR Projects Overview: Automatic Differentiation

- ADOL-C: Package for the automatic differentiation of C and C++ programs.

  Project Manager: Andrea Walther

- CppAD: A tool for differentiation of C++ functions.

  Project Manager: Brad Bell

COIN-OR Projects Overview: Graphs

- GiMPy and GrUMPy: Python packages for visualizing algorithms

  Project Manager: Ted Ralphs

- Cgc: Coin graph class utilities, etc.

  Project Manager: Phil Walton

- LEMON: Library of Efficient Models and Optimization in Networks

  Project Manager: Alpar Juttner

COIN-OR Projects Overview: Miscellaneous

- Djinni: C++ framework with Python bindings for heuristic search

  Project Manager: Justin Goodson
- METSlib: An object oriented metaheuristics optimization framework and toolkit in C++

  Project Manager: Mirko Maischberger
- CoinBazaar: A collection of examples, application codes, utilities, etc.

  Project Manager: Bill Hart
- PFunc: Parallel Functions, a lightweight and portable library that provides C and C++ APIs to express task parallelism

  Project Manager: Prabhanjan Kambadur
- ROSE: Reformulation-Optimization Software Engine, software for performing symbolic reformulations to Mathematical Programs (MP)

  Project Manager: David Savourey
- MOCHA: Matroid Optimization: Combinatorial Heuristics and Algorithms, heuristics and algorithms for multicriteria matroid optimization

  Project Manager: David Hawes

Outline

The COIN-OR Optimization Suite

- Many of the tools in the repository that are focused on solution of mathematical optimization models are inter-related.
- They are built from a common underlying set of tools in a hierarchical fashion using a common build harness.
- The COIN-OR Optimization Suite is an umbrella project that consists of compatible version of all these mutually interoperable projects.
- This suite will be the focus of the remainder of the tutorial.

Modular Structure of the Suite

- 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:

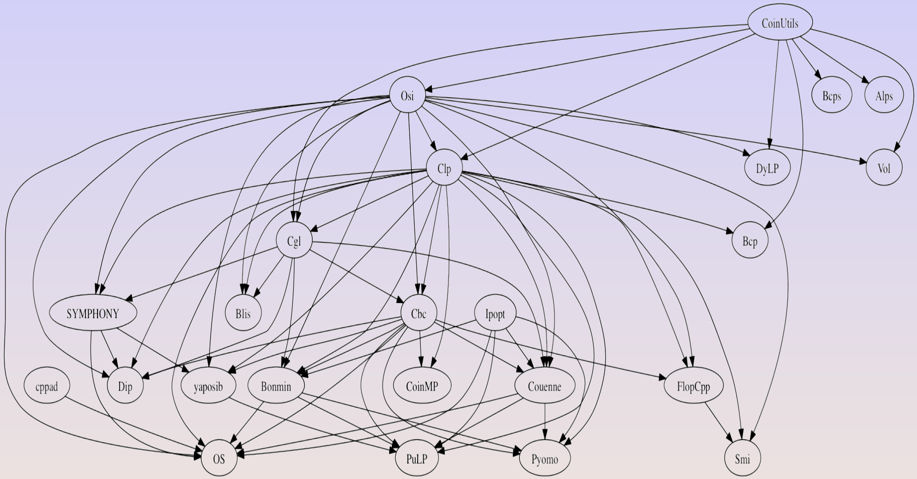
- CBC
- CLP
- CPLEX
- DyLP
- FortMP
- XPRESS-MP
- GLPK
- Mosek
- OSL
- Soplex
- SYMPHONY
- Volume Algorithm

- 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 binaries here:

  `http://www.coin-or.org/download/binary/CoinAll`

- Installers
  - For Windows, there is an automated installer available at the URL above for Visual Studio compatible libraries built with the open source InstallJammer (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—interested?)
  - 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 `CoinAll` source tarballs and zip archives here:

  `http://www.coin-or.org/download/source/CoinAll`
- The recommended way to get source is to use `subversion`, although git will soon be an option as well.
- 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/CoinAll`

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 (see `http://semver.org/`).
  - 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.

Subversion Repo Layout for Project

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

  - Trunk is where development takes place (bleeding edge).
  - Stable versions have wo 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:

Source Tree for Project Xxx Root

```
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:

Source Tree for Project Xxx Subdirectory

```
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.

Monolithic Builds from Source (*nix)

- To check out `CoinAll` (or any other project) and build all required libraries and binaries from source.

Monolithic Build

```
svn co http://projects.coin-or.org/svn/CoinBinary/CoinAll/stable/1.7 \
     CoinAll-1.7
cd CoinAll-1.7
mkdir build
cd build
../configure --enable-gnu-packages -C --prefix=/path/to/install/dir
make -j 2
make test
make install
```

- Note that after building, the examples will be installed with Makefiles in project subdirectories.
- If you move the installed libraries or link to them from a non-COIN binary, you need to set either `LD_LIBRARY_PATH` (Linux) or `DYLD_LIBRARY_PATH` (OS X) to point to the installation directory.

Building on Windows (GNU Autotools)

- Transparently build with either CYGWIN, MinGW, or cl compilers.
- First step is to install either Msys or CYGWIN.
  - For MSys, download the MinGW installer and run it.
  - Add `C:\MinGW\bin` to your `PATH`.
  - Run `mingw-get install msys`.
  - Add `C:\MinGW\msys\1.0\bin` to your `PATH`.
  - Open `bash`.
- 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, pkg-config and probably 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`.
  - Open `bash`.

Building on Windows (GNU Autotools continued)

- 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-liner 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).
  - Run `bash`.
  - Configure with `-enable-msvc` (you can build for a different run-time by, e.g., `-enable-msvc=MT`).
  - To build Python extensions, you should use the Visual Studio compiler and build with `-enable-msvc=MD`.
- To build with the GNU compilers
  - In CYGWIN, just follow the instructions for Linux/OS X.
  - In MSys, you must get also install the MinGW compilers with `mingw-get install gcc g++ gfortran` and configure with `-disable-pkg-config`.

Building on Windows (Visual Studio IDE)

- To build through the Visual Studio IDE, MSVC++ project files are provided.
  - Current standard version of the compiler is v10 (use v9 for Python extensions).
  - Important: We recently switched to using property sheets to save common settings!
  - 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.

Building on OS X

- Getting the compilers you need can be a bit of a pain on OS X, though it is vastly better now than is used to be.
  - The latest versions of OS X come with the clang compiler but no Fortran compiler (the “gcc” command on OS X is really clang).
  - Since clang uses the GNU standard library, gfortran is compatible and can be installed from Homebrew.
  - Older versions of OS X don’t come with any compiler, but you can get gcc with Xcode or in some cases as a separate package (Apple command-line tools).
  - In all cases, gcc/g++ can be obtained through either Homebrew or Macports (I recommend Homebrew).
- The recommended way to build is with Homebrew recipes when they are available (this is a work in progress).
- Otherwise, open a terminal and follow the instructions for Linux.
- Clang will be used by default. If you want, e.g., the gcc provided by Homebrew, you need to specify that with `CC=gcc CXX=g++`.

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 Projects from Source (*Nix)

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

Tweaking a Single Library

```
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 checkout stable/X.X
git checkout releases/X.X.X
```

- There isn't a very convenient way to replicate a checkout including externals in git.
- Right now, you need to check out the individual projects, then build and install them in sequence.

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.

Class Exercise: Download and Install COIN

End of Part 1!

Questions?