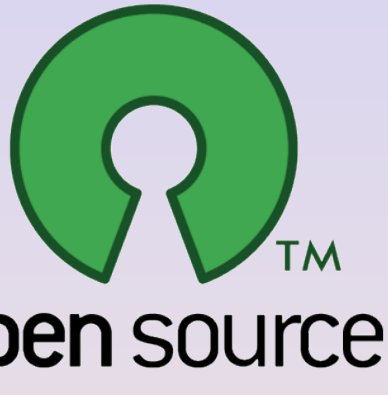
Build and Test

The COIN-OR Way

Ted Ralphs

LEHIGH
UNIVERSITY

COR@L
COMPUTATIONAL OPTIMIZATION RESEARCH AT LEHIGH



COIN fORgery: Developing Open Source Tools for OR

Institute for Mathematics and Its Applications, Minneapolis, MN

Outline

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 (Msys2, WSL, 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.

Building on Linux/WSL

Use the `get.dependencies` script, a `bash` script that replaces the externals mechanism and the monolithic build mechanism.

- The `fetch` command gets dependencies (using `git` or `svn` for mirrored projects) and optionally downloads third-party codes.
- The `build` command builds dependencies in order, pre-installs them in the build directory, and optionally runs unit tests.
- The `install` command installs all code in the final location.

- Running `fetch` again updates existing code.
- Running `build` againcontinues/rebuilds with cached options (reconfigure can be foreced if desired).
- To obtain the script, do

```
git clone https://github.com/coin-or-tools/BuildTools
svn co https://projects.coin-or.org/svn/BuildTools/trunk BuildTools
```

Building on Windows (GNU Autotools)

- Transparently build with either CYGWIN, MinGW, or cl compilers.
- First step is to either install MSys2 or enable WSL.
  - For MSys2, download and run the MSys2 installer.
  - Add `C:\MSys64\usr4\bin` to your `PATH`.
  - Run `bash`
  - Run `pacman -S git svn make wget tar patch dos2unix mingw-w64-i686-gcc mingw-w64-i686-gcc-fortran mingw-w64-x86_64-gcc mingw-w64-x86_64-gcc-fortran`
  - Add `/mingw64/bin` to your `PATH`.
  - Run `get.dependencies fetch` and `get.dependencies build`.

Building on Windows (GNU Autotools continued)

- To build with the Visual Studio compiler
  - 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).
  - 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, just follow the instructions for Linux.

Building on Windows (Visual Studio IDE)

- To build through the Visual Studio IDE, MSVC++ project files are provided, but may be out of date.
  - Current standard version of the compiler is v10, but upgrading to your version should work.
  - Important: We use 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

- OS X comes with the clang compiler but no Fortran compiler.
- Since clang uses the GNU standard library, gfortran is compatible and can be installed from Homebrew.
- gcc/g++ can also be obtained through 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++`.

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.

Outline

Overall Goals

- Basic goals
  - Ensure codes continue to work as expected.
  - Ensure portability across different platforms.
  - Build and deploy working, partable binaries automatically.
- There is a working infrastructure in place for accomplishing this, but it needs additional work.
- At the moment, the infrastructure is focused on the C++ projects.
- There are also a number of Python projects that either need testing or could be used for testing.

Current Toolbox

- Git and Github
- Travis
- Appveyor
- Bintray
- Homegrown unit tests
- Homegrown build scripts
  - Autotools
  - Some homebrew
  - Some conda

Unit Testing Paradigm

- There is no real standard for how unit testing is done across all projects.
- All projects within the Optimization Suite do have substantial unit tests.
- Most unit tests consist of simple asserts on various expected behaviors.
- The result is that the unit test chokes and dies as soon one test fails, which is not particularly helpful.
- There was a serious effort to improve the unit testing, starting in OSI, a few years ago.
- This has not been completed.

Aspirations

- Bring unit testing into a single unified and modern framework.
- Make it easy to add and execute tests.
- What is the right framework?
- Google Test look as good an option as any.
- Converting existing unit tests is a manual process, but many hands make light work.

Example: CoinUtils

```
/*
   Explicitly request gap on this matrix, so we can see it propagate
   in subsequent copy & assignment.
*/
CoinPackedMatrix pm(false,minor,major,numels,elem,ind,starts,lens,
                    .25,.25);

assert( elem!=NULL );
assert( ind!=NULL );
assert( starts!=NULL );
assert( lens!=NULL );

delete[] elem;
delete[] ind;
delete[] starts;
delete[] lens;

assert( eq(pm.getExtraGap(),.25) );
assert( eq(pm.getExtraMajor(),.25) );
assert( !pm.isColOrdered() );
assert( pm.getNumElements()==numels );
assert( pm.getNumCols()==minor );
assert( pm.getNumRows()==major);
assert( pm.getSizeVectorStarts()==major+1 );
assert( pm.getSizeVectorLengths()==major );
```

Example: OSI

```
  OsiColCut r;

  // Test setting/getting bounds
  r.setLbs( ne, inx, el );
  r.setEffectiveness(222.);
  OSIUNITTEST_ASSERT_ERROR(r.lbs().getNumElements() == ne, return, "osicolcut", "se
  bool bounds_ok = true;
  for ( int i=0; i<ne; i++ ) {
   bounds_ok &= r.lbs().getIndices()[i] == inx[i];
   bounds_ok &= r.lbs().getElements()[i] == el[i];
  }
  OSIUNITTEST_ASSERT_ERROR(bounds_ok, {}, "osicolcut", "setting bounds");
  OSIUNITTEST_ASSERT_ERROR(r.effectiveness() == 222.0, {}, "osicolcut", "setting bo

  r.setUbs( ne3, inx3, el3 );
  OSIUNITTEST_ASSERT_ERROR(r.ubs().getNumElements() == 0, {}, "osicolcut", "settin
  OSIUNITTEST_ASSERT_ERROR(r.ubs().getIndices()  == NULL, {}, "osicolcut", "settin
  OSIUNITTEST_ASSERT_ERROR(r.ubs().getElements() == NULL, {}, "osicolcut", "settin
}
```

Running the Unit Test

- Support for unit testing can be easily added to the build setup provided by the `BuildTools`.
- Simply add desired tests to the `test` target in `Makefile.am` in the `test` subdirectory.

```
unittestflags =
if COIN_HAS_SAMPLE
  unittestflags += -mpsDir=`$(CYGPATH_W) $(SAMPLE_DATA)`
endif
if COIN_HAS_NETLIB
  unittestflags += -netlibDir=`$(CYGPATH_W) $(NETLIB_DATA)`\
                   -testModel=adlittle.mps
endif

test: unitTest$(EXEEXT)
./unitTest$(EXEEXT) $(unittestflags)
```

- Then either invoke `make test` directly or run `get.dependencies build -test`.

Move To Google Test (or Similar)?

It looks relatively starightfoward, though a lot of manual work, to move to Google Test.

```
#include "gtest/gtest.h"

TEST(FactorialTest, HandlesPositiveInput) {
  EXPECT_EQ(Factorial(1), 1);
  EXPECT_EQ(Factorial(2), 2);
  EXPECT_EQ(Factorial(3), 6);
  EXPECT_EQ(Factorial(8), 40320);
}
```

Outline

Automating Build and Test

- The automated build and test in COIN-OR uses
  - Travis (see https://travis-ci.org/coin-or/)
  - Appveyor (see https://ci.appveyor.com)
  - Bintray (https://bintray.com/coin-or/download)
- Enabling these services is as easy as authenticating with your Github account and adding a configuration file to your repo.
- To enable for projects in the COIN-OR organization, just ask me!
- Configuration files are in YAML format.

Travis YAML File (Matrix)

```
language: cpp
env:
  global:
    - secure: "WEPiGQTAVh02Fc5Qxvj2VBZoWBgNjjB6w5MdNwg6NtzFKhbhxB1OqfaVE7EJV1Cw9mf1k5

matrix:
  include:
    - os: linux
      addons:
        apt:
          packages:
            - gfortran
    - os: osx
      osx_image: xcode8.2
      env: OSX=10.12
      compiler: clang
    - os: osx
      osx_image: xcode8
      env: OSX=10.11
      compiler: clang
    - os: osx
      osx_image: xcode6.4
      env: OSX=10.10
      compiler: clang
  allow_failures:
    - os: osx
```

Travis YAML File (Build and Test

```
before_script:
  - if [[ "$TRAVIS_OS_NAME" == "osx" ]]; then  \
    export PLATFORM=osx$OSX-x86_64-clang`clang -dumpversion`; fi
  - if [[ "$TRAVIS_OS_NAME" == "osx" ]]; then brew update; brew install bash \
    gcc; brew link --overwrite gcc; gfortran --version; fi
  - if [[ "$TRAVIS_OS_NAME" == "linux" ]]; then \
    export PLATFORM=linux-x86_64-gcc`gcc -dumpversion`; fi
  - git clone https://github.com/coin-or-tools/BuildTools
  - bash BuildTools/get.dependencies.sh fetch > /dev/null

script:
  - bash BuildTools/get.dependencies.sh build --verbosity=2 --test

after_script:
  - if [ $TRAVIS_BRANCH = "master" ]; then export VERSION=trunk; else \
    export VERSION=`echo $TRAVIS_BRANCH | cut -d "/" -f 2`; fi
  - export PROJECT=`echo $TRAVIS_REPO_SLUG | cut -d "/" -f 2`
  - export TGZ_FILE=$PROJECT-$VERSION-$PLATFORM.tgz
  - echo $TGZ_FILE
  - tar -czvf $TGZ_FILE build/lib/* build/bin/* build/include/* build/share/* \
    README.md INSTALL LICENSE Cbc/AUTHORS
  - curl -T $TGZ_FILE -utkralphs:$BINTRAY_API -H "X-Bintray-Publish:1" -H \
    "X-Bintray-Override:1" \
    https://api.bintray.com/content/coin-or/download/$PROJECT/$VERSION/$TGZ_FILE
```

Appveyor YAML File (Matrix)

```
#init:
#  - ps: iex ((new-object net.webclient).DownloadString('https://raw.githubuserconter

platform:
  - x64

environment:
  global:
    BINTRAY_API:
      secure: a9n4jf90wlFCdaYa6fOmYxsF97ur2dnK8Ys3gn5R90JBzTDq6cD2GlEwmmts75mq
    BINTRAY_USERNAME: tkralphs
  matrix:
    - ARCH: win32-msvc9
      HOST_ARCH_ARG: --enable-msvc=MD
      ADD_PATH: /mingw64/bin
    - ARCH: win32-msvc12
      HOST_ARCH_ARG: --enable-msvc
      ADD_PATH: /mingw64/bin
    - ARCH: win32-msvc14
      HOST_ARCH_ARG: --enable-msvc
      ADD_PATH: /mingw64/bin
    - ARCH: x86_64-w64-mingw32
      HOST_ARCH_ARG: --host=x86_64-w64-mingw32
      ADD_PATH: /mingw64/bin
    - ARCH: i686-w64-mingw32
      HOST_ARCH_ARG: --host=i686-w64-mingw32
      ADD_PATH: /mingw32/bin
```

Appveyor YAML File (Build and Test)

```
install:
  - for /f "delims=" %%i in ('C:\msys64\usr\bin\bash -lc "if [ $APPVEYOR_REPO_BRANCH = 'master' ]; then echo 'trunk';
  - echo %VERSION%
  - IF %ARCH%==win32-msvc9 (CALL C:\"Program Files (x86)\Microsoft Visual Studio 9.0\VC\vcvarsall.bat")
  - IF %ARCH%==win32-msvc12 (CALL C:\"Program Files (x86)\Microsoft Visual Studio 12.0\VC\vcvarsall.bat")
  - IF %ARCH%==win32-msvc14 (CALL C:\"Program Files (x86)\Microsoft Visual Studio 14.0\VC\vcvarsall.bat")
  - C:\msys64\usr\bin\bash -lc ""

build_script:
  - C:\msys64\usr\bin\bash -lc "cd $APPVEYOR_BUILD_FOLDER; git clone https://github.com/coin-or-tools/BuildTools"
  - C:\msys64\usr\bin\bash -lc "cd $APPVEYOR_BUILD_FOLDER; BuildTools/get.dependencies.sh fetch --no-third-party"
  - C:\msys64\usr\bin\bash -lc "cd $APPVEYOR_BUILD_FOLDER; export PATH=$ADD_PATH:$PATH; \
    BuildTools/get.dependencies.sh build --build=x86_64-w64-mingw32 $HOST_ARCH_ARG --verbosity=2 --test"

after_build:
  - 7z a %APPVEYOR_PROJECT_NAME%-%VERSION%-%ARCH%.zip %APPVEYOR_BUILD_FOLDER%\build\bin
  - 7z a %APPVEYOR_PROJECT_NAME%-%VERSION%-%ARCH%.zip %APPVEYOR_BUILD_FOLDER%\build\include
  - 7z a %APPVEYOR_PROJECT_NAME%-%VERSION%-%ARCH%.zip %APPVEYOR_BUILD_FOLDER%\build\lib
  - 7z a %APPVEYOR_PROJECT_NAME%-%VERSION%-%ARCH%.zip %APPVEYOR_BUILD_FOLDER%\build\share
  - 7z a %APPVEYOR_PROJECT_NAME%-%VERSION%-%ARCH%.zip %APPVEYOR_BUILD_FOLDER%\README.md
  - 7z a %APPVEYOR_PROJECT_NAME%-%VERSION%-%ARCH%.zip %APPVEYOR_BUILD_FOLDER%\LICENSE
  - 7z a %APPVEYOR_PROJECT_NAME%-%VERSION%-%ARCH%.zip %APPVEYOR_BUILD_FOLDER%\INSTALL
  - 7z a %APPVEYOR_PROJECT_NAME%-%VERSION%-%ARCH%.zip %APPVEYOR_BUILD_FOLDER%\Cbc\AUTHORS
  - curl -T %APPVEYOR_PROJECT_NAME%-%VERSION%-%ARCH%.zip
    -utkralphs:%BINTRAY_API% -H "X-Bintray-Publish:1" \
    -H "X-Bintray-Override:1" https://api.bintray.com/content/coin-or/download/%APPVEYOR_PROJECT_NAME%/%VERSION%/%APPV

#on_finish:
#  - ps: $blockRdp = $true; iex ((new-object
#net.webclient).DownloadString('https://raw.githubusercontent.com/appveyor/ci/master/scripts/enable-rdp.ps1'))
```