An Introduction to the COIN-OR Optimization Suite:
Open Source Tools for Building and Solving Optimization Models

TED RALPHS
ISE Department
COR@L Lab
Lehigh University
ted@lehigh.edu

Optimization Days, Montreal, May 7, 2013
I’ll touch on a lot of things and can drill down if there’s interest.

There is an inevitable bias towards things that I work on.

I’m going to talk about the work of lots of different people and will inevitably miss some attributions.

The talk proceeds from general high level tools down to lower level tools, feel free to leave when you’ve seen enough.

I’ll try to focus on the “not-so-obvious” bits.

Please ask questions! I may or may not be able to answer them.

Let’s Go!
Outline

1 Introduction to COIN
   - COIN-OR Foundation
   - Overview of Projects

2 Overview of Optimization Suite
   - Installing the COIN Optimization Suite
   - Documentation and Support

3 Entry Points
   - Modeling Systems
   - Python Tools
   - Command-line Tools
   - Building Applications

4 Advanced Development
   - SYMPHONY
   - DIP
   - CHiPPS
   - Working with Source
Outline

1. Introduction to COIN
   - COIN-OR Foundation
   - Overview of Projects

2. Overview of Optimization Suite
   - Installing the COIN Optimization Suite
   - Documentation and Support

3. Entry Points
   - Modeling Systems
   - Python Tools
   - Command-line Tools
   - Building Applications

4. Advanced Development
   - SYMPHONY
   - DIP
   - CHiPPS
   - Working with Source
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 has now grown to be self-sustaining and was spun off as a nonprofit educational foundation in the U.S. several years 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.

See [www.coin-or.org](http://www.coin-or.org) for more information.

T.K. Ralphs (Lehigh University)

COIN-OR

7 May, 2013
The COIN-OR Foundation is governed by two boards.

**Strategic Leadership Board**
- Matt Saltzman (President)
- Lou Hafer (Secretary)
- Randy Kiefer (Treasurer)
- Ted Ralphs (TLC Rep)
- Bill Hart
- Kevin Furman
- Alan King

**Technical Leadership Council**
- Ted Ralphs (Chair)
- Kipp Martin
- Stefan Vigerske
- John Siirola
- Matthew Galati
- Haroldo Santos

- 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?

Hello COIN-OR Foundation, Inc,

You received a payment from [name] for Donation

Customer details

Customer name: [name]
Customer email: [email]
Profile ID: [ID]
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’s Happening at COIN

- 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

- Re-launching Web site with many new features
  - Forums
  - Social integration, single sign-on (OpenID)
  - Support for git
  - Individual project Web sites
- Installers
- RPMs and .debs
- Modeling tools
- Python support
- New versions of most tools \( \Leftarrow \) due out imminently!
- And more...
We currently have 50+ projects and more are being added all the time.

Most projects are now licensed under the EPL (very permissive).

COIN 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 has various utilities for reading/building/manipulating/preprocessing optimization models and getting them into solvers.

COIN has overarching frameworks that support implementation of broad algorithm classes.

- Parallel search
- Branch and cut (and price)
- Decomposition-based algorithms
One of the most exciting developments of recent years is the number of high-level tools available to access COIN solvers. COIN isn’t just for breakfast anymore!

- Python-based modeling languages
- Spreadsheet modeling (!)
- Commercial modeling languages
- Matlab
- R
- Sage (?)
- Julia
- ...

COIN isn’t just for breakfast anymore!
Outline

1. Introduction to COIN
   - COIN-OR Foundation
   - Overview of Projects

2. Overview of Optimization Suite
   - Installing the COIN Optimization Suite
   - Documentation and Support

3. Entry Points
   - Modeling Systems
   - Python Tools
   - Command-line Tools
   - Building Applications

4. Advanced Development
   - SYMPHONY
   - DIP
   - CHiPPS
   - Working with Source
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
**Ipopt**: Interior Point OPTimizer implements interior point methods for solving 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 for support vector machines, solver for the sparse inverse covariance problem.

*Project Manager*: Katya Scheinberg
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

- **COOPR**: 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
**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
**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
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

1. Introduction to COIN
   - COIN-OR Foundation
   - Overview of Projects

2. Overview of Optimization Suite
   - Installing the COIN Optimization Suite
   - Documentation and Support

3. Entry Points
   - Modeling Systems
   - Python Tools
   - Command-line Tools
   - Building Applications

4. Advanced Development
   - SYMPHONY
   - DIP
   - CHiPPS
   - Working with Source
Many of the tools mentioned interoperate by using the configuration and build utilities provided by the BuildTools project.

The interoperable suite of tools for optimization will be the focus of the remainder of the tutorial.

The BuildTools project provides build infrastructure for

- MS Windows (CYGWIN, MINGW, and Visual Studio)
- Linux
- Mac OS X

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 CoinAll project is an über-project that includes compatible version of all mutually interoperable projects.

The easiest way to use multiple COIN projects is simply download and install the latest version of CoinAll (1.7 is due out imminently).

The TestTools project is the focal point for testing of COIN code.
You can download CoinAll binaries here:

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

About version numbers

- COIN numbers versions by a standard major, minor, release scheme.
- All version within a major.minor series are compatible.
- All versions within a major series are backwards compatible.

The CoinBinary project is a long-term effort to provide pre-built binaries for popular platforms.

- We now have (beta) cross-platform installers built with the open source InstallJammer.
- We are in the process of being approved for inclusion in the Fedora distribution (RPM).
- COIN can already be installed with apt-get on Ubuntu, but these versions are quite old now.

Other ways of obtaining COIN include downloading it through a number of modeling language front-ends (more on this later).
Installing from 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 may 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.

- 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
Outline

1 Introduction to COIN
   - COIN-OR Foundation
   - Overview of Projects

2 Overview of Optimization Suite
   - Installing the COIN Optimization Suite
   - Documentation and Support

3 Entry Points
   - Modeling Systems
   - Python Tools
   - Command-line Tools
   - Building Applications

4 Advanced Development
   - SYMPHONY
   - DIP
   - CHiPPS
   - Working with Source
- 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.
Outline

1. Introduction to COIN
   - COIN-OR Foundation
   - Overview of Projects

2. Overview of Optimization Suite
   - Installing the COIN Optimization Suite
   - Documentation and Support

3. Entry Points
   - Modeling Systems
   - Python Tools
   - Command-line Tools
   - Building Applications

4. Advanced Development
   - SYMPHONY
   - DIP
   - CHiPPS
   - Working with Source
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**
  - **Coopr** (Pyomo, PySP, SUCASA)
  - **PuLP/Dippy** (Decomposition-based modeling)
  - **CyLP** (provides API-level interface)
  - **yaposib** (OSI bindings)

- **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)
  - **R** (RSymphony Plug-in)
  - **Matlab** (OPTI)
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
To use OS to call solvers in AMPL, you specify the `OSAmplClient` as the solver.

```plaintext
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.

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

1. Introduction to COIN
   - COIN-OR Foundation
   - Overview of Projects

2. Overview of Optimization Suite
   - Installing the COIN Optimization Suite
   - Documentation and Support

3. Entry Points
   - Modeling Systems
   - Python Tools
     - Command-line Tools
     - Building Applications

4. Advanced Development
   - SYMPHONY
   - DIP
   - CHiPPS
   - Working with Source
Drinking the Python Kool-Aid

I learned it last night! Everything is so simple!
Hello world is just print "Hello, world!"

I dunno...
Dynamic typing?
Whitespace?
Come join us!
Programming is fun again!
It's a whole new world up here!
But how are you flying?

I just typed:
import antigravity
That's it?

...I also sampled everything in the medicine cabinet for comparison.

But I think this is the Python.
Why Python?

Singing the Praises

- As with many high-level languages, development is quick and painless (relative to C++!)
- Python is popular in many disciplines and there is a dizzying array of packages available.
- Python’s syntax is very clean and naturally adaptable to expressing mathematical programming models.
- Python has the primary data structures necessary to build and manipulate models built in.

- There has been a very strong movement in recent years toward the adoption of Python as the high-level language of choice for (discrete) optimizers.
- For these reasons and more, Sage is quickly emerging as a very capable open-source alternative to Matlab.
- Python does have one major downside: it can be very slow.
- One solution is to write extensions in C/C++: COIN!
- Go and Julia are faster alternatives that retain many of Python’s advantages.
Python is dynamically typed.
No memory allocation or freeing, no variable declarations
Indentation has a syntactic meaning: no curly braces and good formatting is required!
Code is usually easy to read “in English” (keywords like `is`, `not`, and `in`).
Everything is a pointer to an object: functions, classes, variables,...
Everything can be “printed.”
Built-in data structures:
- Lists (dynamic arrays)
- Dictionaries (hash tables)
- Sets
Easy to define new data types via classes and re-definition of basic operators (magic methods).
Light-weight inheritance mechanism for customizing classes.
Extremely flexible mechanism for passing function arguments.
PuLP (Stu Mitchell)

- A modeling language for expressing linear models in Python.
- Similar to other algebraic modeling languages but with the power of Python.
- Let’s see an example.
Example: Facility Location Problem

- We have \( n \) locations and \( m \) customers to be served from those locations.
- There is a fixed cost \( c_j \) and a capacity \( W_j \) associated with facility \( j \).
- There is a cost \( d_{ij} \) and demand \( w_{ij} \) for serving customer \( i \) from facility \( j \).
- We have two sets of binary variables.
  - \( y_j \) is 1 if facility \( j \) is opened, 0 otherwise.
  - \( x_{ij} \) is 1 if customer \( i \) is served by facility \( j \), 0 otherwise.

### Capacitated Facility Location Problem

\[
\begin{align*}
\text{min} & \quad \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} \\
\text{s.t.} & \quad \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \\
& \quad \sum_{i=1}^{m} w_{ij} x_{ij} \leq W_j \quad \forall j \\
& \quad x_{ij} \leq y_j \quad \forall i, j \\
& \quad x_{ij}, y_j \in \{0, 1\} \quad \forall i, j
\end{align*}
\]
from products import REQUIREMENT, PRODUCTS
from facilities import FIXED_CHARGE, LOCATIONS, CAPACITY

prob = LpProblem("Facility_Location")

ASSIGNMENTS = [(i, j) for i in LOCATIONS for j in PRODUCTS]
assign_vars = LpVariable.dicts("x", ASSIGNMENTS, 0, 1, LpBinary)
use_vars = LpVariable.dicts("y", LOCATIONS, 0, 1, LpBinary)

prob += lpSum(use_vars[i] * FIXED_COST[i] for i in LOCATIONS)

for j in PRODUCTS:
    prob += lpSum(assign_vars[(i, j)] for i in LOCATIONS) == 1

for i in LOCATIONS:
    prob += lpSum(assign_vars[(i, j)] * REQUIREMENT[j] for j in PRODUCTS) <= CAPACITY * use_vars[i]

prob.solve()
PuLP Basics: Facility Location Example

# The requirements for the products
REQUIREMENT = {
    1 : 7,
    2 : 5,
    3 : 3,
    4 : 2,
    5 : 2,
}

# Set of all products
PRODUCTS = REQUIREMENT.keys()
PRODUCTS.sort()

# Costs of the facilities
FIXED_COST = {
    1 : 10,
    2 : 20,
    3 : 16,
    4 : 1,
    5 : 2,
}
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
from products import REQUIREMENT, PRODUCTS
from facilities import FIXED_CHARGE, LOCATIONS, CAPACITY

prob = dippy.DipProblem("Facility_Location")

ASSIGNMENTS = [(i, j) for i in LOCATIONS for j in PRODUCTS]
assign_vars = LpVariable.dicts("x", ASSIGNMENTS, 0, 1, LpBinary)
use_vars = LpVariable.dicts("y", LOCATIONS, 0, 1, LpBinary)

prob += lpSum(use_vars[i] * FIXED_COST[i] for i in LOCATIONS)

for j in PRODUCTS:
    prob += lpSum(assign_vars[(i, j)] for i in LOCATIONS) == 1

for i in LOCATIONS:
    prob.relaxation[i] += lpSum(assign_vars[(i, j)] * REQUIREMENT[j]
               for j in PRODUCTS) <= CAPACITY * use_vars[i]

dippy.Solve(prob, {doPriceCut:1})
SolverStudio (Andrew Mason)

- Spreadsheet optimization has had a (deservedly) bad reputation for many years.
- SolverStudio will change your mind about that!
- SolverStudio provides a full-blown modeling environment inside a spreadsheet.
  - Edit and run the model.
  - Populate the model from the spreadsheet.
Coopr and Pyomo

- An algebraic modeling language in Python similar to PuLP.
- More powerful, includes support for nonlinear modeling.
- Coopr also include PySP for stochastic Programming.

*Developers*: Bill Hart, David Woodruff, John Sirola, and others at Sandia National Labs.
CyLP: Low-level API for Cbc/Clp

- A lower-level modeling language for accessing details of the algorithms and low-level parts of the API.
- Clp
  - Pivot-level control of algorithm in Clp.
  - Access to fine-grained results of solve.
- Cbc
  - Python class for cut generators

*Developers*: Mehdi Towhidi and Dominique Orban
lp = CyClpSimplex()
x = lp.addVariable('x', numVars)
lp += x_u >= x >= 0
lp += (A * x <= b if cons_sense == '<=' else A * x >= b)
lp.objective = -c * x if obj_sense == 'Max' else c * x
lp.primal(startFinishOptions = 1)
numCons = len(b)
print 'Current solution is', lp.primalVariableSolution['x']
numAllVars = len(lp.primalVariableSolutionAll)

tableaux = np.zeros(shape = (numAllVars, numCons))
for i in range(numAllVars):
    lp.getBInvACol(i, tableaux[i,:])
tableaux = tableaux.transpose()
rhs = tableaux[:,numVars:]*np.matrix(b).transpose()
GiMPy (with Aykut Bulut)

- A graph class for Python 2.6.
- Builds, displays, and saves graphs (many options)
- Focus is on visualization of well-known graph algorithms.
  - Priority in implementation is on clarity of the algorithms.
  - Efficiency is not the goal (though we try to be as efficient as we can).

```python
from gimpy import Graph
if __name__=='__main__':
    g = Graph(display='pygame')
    g.add_edge(0,1)
    g.add_edge(1,2)
    g.add_edge(3,4)
    g.display()
    g.search(0)
```
GIMPy Example
The following problem/algorithm pairs with similar visualization options exist.

- **Graph Search:**
  - BFS
  - DFS
  - Prim’s
  - Component Labeling,
  - Dijkstra’s
  - Topological Sort

- **Shortest path:** Dijkstra’s, Label Correcting

- **Maximum flow:** Augmenting Path, Preflow Push

- **Minimum spanning tree:** Prim’s Algorithm, Kruskal Algorithm

- **Minimum Cost Flow:** Network Simplex, Cycle Canceling

- **Data structures:** Union-Find (quick union, quick find), Binary Search Tree, Heap
GiMPy Tree

- Tree class derived from Graph class.
- BinaryTree class derived from Tree class.
- Has binary tree specific API and attributes.
Visualizations for solution methods for linear models.

- Branch and bound
- Cutting plane method

**BBTree derived from GiMPy Tree.**
- Reads branch-and-bound data either dynamically or statically.
- Builds dynamic visualizations of solution process.
- Includes a pure Python branch and bound implementation.

**Polyhedron2D derived from pypolyhedron.**
- Can construct 2D polyhedra defined by generators or inequalities.
- Displays convex hull of integer points.
- Can produce animations of the cutting plane method.

GrUMPy is an expansion and continuation of the BAK project (Brady Hunsaker and Osman Ozaltin).
GrUMPy provides four visualizations of the branch and bound process. Can be used dynamically or statically with any instrumented solver.

- BB tree
- Histogram
- Scatter plot
- Incumbent path
GrUMPy Branch and Bound Tree

Figure: BB tree generated by GrUMPy
Figure: BB histogram generated by GrUMPy
Figure: Scatter plot generated by GrUMPy
Figure: Incumbent path generated by GrUMPy
GrUMPy: Polyhedron2D

```python
fig = plt.figure()
ax = fig.add_subplot(111)
ax.grid()
if points is not None:
    p = Polyhedron2D(points = points, rays = rays)
else:
    p = Polyhedron2D(A = A, b = b)
p.draw(ax, color = 'blue', linestyle = 'solid')
ax.set_xlim(p.plot_min[0], p.plot_max[0])
ax.set_ylim(p.plot_min[1], p.plot_max[1])
pI = p.make_integer_hull()
pI.draw(ax, color = 'red', linestyle = 'dashed')
if c is not None:
    add_line(ax, c, obj_val, p.plot_max - [0.2, 0.2], p.plot_min +
    linestyle = 'dashed')
plt.show()
```
GrUMPy: Polyhedron2D
GrUMPy: Polyhedron2D
GrUMPy: Polyhedron2D
GrUMPy: Polyhedron2D
GrUMPy: Polyhedron2D
GrUMPy: Polyhedron2D
Outline

1. Introduction to COIN
   - COIN-OR Foundation
   - Overview of Projects

2. Overview of Optimization Suite
   - Installing the COIN Optimization Suite
   - Documentation and Support

3. Entry Points
   - Modeling Systems
   - Python Tools
   - Command-line Tools
   - Building Applications

4. Advanced Development
   - SYMPHONY
   - DIP
   - CHiPPS
   - Working with Source
Interactive Shells

A number of projects provide interactive shells (SYMPHONY, CLP, Cbc, OS)

```
~/COIN/trunk/build/bin > ./symphony
== Welcome to the SYMPHONY MILP Solver
== Copyright 2000-2011 Ted Ralphs and others
== All Rights Reserved.
== Distributed under the Eclipse Public License 1.0
== Version: Trunk (unstable)
== Build Date: Mar 16 2013
== Revision Number: 2068

***** 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 in incon). Note that shells are more capable when readline and history are available.
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
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
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.
Outline

1. Introduction to COIN
   - COIN-OR Foundation
   - Overview of Projects

2. Overview of Optimization Suite
   - Installing the COIN Optimization Suite
   - Documentation and Support

3. Entry Points
   - Modeling Systems
   - Python Tools
   - Command-line Tools
   - Building Applications

4. Advanced Development
   - SYMPHONY
   - DIP
   - CHiPPS
   - Working with Source
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.
Using SYMPHONY API

```c
#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
**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 --libs cbc
  ```

- To determine the flags that should be given to the compiler, the command is
  
  ```
  pkg-config --cflags 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
  ```
The `pkg-config` command can be used to vastly simplify the Makefiles used to build project that link with COIN.

```bash
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`. 
Libtool Versioning (Shared Libraries)

- Libtool 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.
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

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

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

T.K. Ralphs  (Lehigh University)  COIN-OR  7 May, 2013  97 / 134
Building Blocks: Open Solver Interface

- Uniform API for a variety of solvers: CBC, CLP, CPLEX, DyLP, FortMP, GLPK, Mosek, OSL, Soplex, SYMPHONY, the Volume Algorithm, XPRESS-MP supported to varying degrees.

- 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: Open Solver Interface

- Uniform API for a variety of solvers: CBC, CLP, CPLEX, DyLP, FortMP, GLPK, Mosek, OSL, Soplex, SYMPHONY, the Volume Algorithm, XPRESS-MP supported to varying degrees.
- 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.
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
Building Blocks: 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

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

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

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

Step 4: Build solver-specific model instance

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

Step 5: Solve the problem.

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

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

- Put some variables in

  ```cpp
  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.
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.
def solve_subproblem(prob, index, redCosts, convexDual):
    ...
    return knapsack01(obj, weights, CAPACITY)
def knapsack01(obj, weights, capacity):
    ...
    return solution
def first_fit(prob):
    ...
    return bvs
prob.init_vars = first_fit
def choose_branch(prob, sol):
    ...
    return ([], down_branch_ub, up_branch_lb, [])
def generate_cuts(prob, sol):
    ...
    return new_cuts
def heuristics(prob, xhat, cost):
    ...
    return sols
dippy.Solve(prob, {'doPriceCut': '1'})
Outline

1 Introduction to COIN
   - COIN-OR Foundation
   - Overview of Projects

2 Overview of Optimization Suite
   - Installing the COIN Optimization Suite
   - Documentation and Support

3 Entry Points
   - Modeling Systems
   - Python Tools
   - Command-line Tools
   - Building Applications

4 Advanced Development
   - SYMPHONY
   - DIP
   - CHiPPS
   - Working with Source
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 (since 1994)
- Biobjective MIP
- Warm starting for MIP
- Sensitivity analysis for MIP

SYMPHONY Applications

- TSP/VRP
- Set Partitioning Problem
- Mixed Postman Problem
- Capacitated Node Routing
- Multicriteria Knapsack
Outline

1. Introduction to COIN
   - COIN-OR Foundation
   - Overview of Projects

2. Overview of Optimization Suite
   - Installing the COIN Optimization Suite
   - Documentation and Support

3. Entry Points
   - Modeling Systems
   - Python Tools
   - Command-line Tools
   - Building Applications

4. Advanced Development
   - SYMPHONY
   - DIP
   - CHiPPS
   - Working with Source
DIP Framework

DIP is a software framework that provides a virtual sandbox for testing and comparing various decomposition-based bounding methods.

- It’s difficult to compare variants of decomposition-based algorithms.
- The method for separation/optimization over a given relaxation is the primary custom component of any of these algorithms.
- **DIP** abstracts the common, generic elements of these methods.
  - **Key:** The user defines methods in the space of the compact formulation.
  - The framework takes care of reformulation and implementation for all variants.
The Cutting Plane Method (CP) iteratively builds an *outer* approximation of $P'$ by solving a cutting plane generation subproblem.
Traditional Decomposition Methods

The **Cutting Plane Method (CP)** iteratively builds an *outer* approximation of $\mathcal{P}'$ by solving a cutting plane generation subproblem.

The **Dantzig-Wolfe Method (DW)** iteratively builds an *inner* approximation of $\mathcal{P}'$ by solving a column generation subproblem.
Traditional Decomposition Methods

The **Cutting Plane Method (CP)** iteratively builds an *outer* approximation of $\mathcal{P}'$ by solving a cutting plane generation subproblem.

![Diagram of Cutting Plane Method](image1)

The **Dantzig-Wolfe Method (DW)** iteratively builds an *inner* approximation of $\mathcal{P}'$ by solving a column generation subproblem.

![Diagram of Dantzig-Wolfe Method](image2)

The **Lagrangian Method (LD)** iteratively solves a Lagrangian relaxation subproblem.

![Diagram of Lagrangian Method](image3)
The **LP bound** is obtained by optimizing over the intersection of two explicitly defined polyhedra.

\[ z_{LP} = \min_{x \in \mathbb{R}^n} \{ c^T x \mid x \in Q^\prime \cap Q^{\prime\prime} \} \]

The **decomposition bound** is obtained by optimizing over the intersection of one explicitly defined polyhedron and one implicitly defined polyhedron.

\[ z_{CP} = z_{DW} = z_{LD} = z_D = \min_{x \in \mathbb{R}^n} \{ c^T x \mid x \in P^\prime \cap Q^{\prime\prime} \} \geq z_{LP} \]

Traditional decomposition-based bounding methods contain two primary steps

- **Master Problem**: Update the primal/dual solution information.
- **Subproblem**: Update the approximation of \( P^\prime \): \( SEP(x, P^\prime) \) or \( OPT(c, P^\prime) \).

**Integrated decomposition methods** further improve the bound by considering two implicitly defined polyhedra whose descriptions are iteratively refined.

- **Price and Cut (PC)**
- **Relax and Cut (RC)**
- **Decompose and Cut (DC)**
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
Automatic Structure Detection

- For unstructured problems, block structure may be detected automatically.
- This is done using hypergraph partitioning methods.
- We map each row of the original matrix to a hyperedge and the nonzero elements to nodes in a hypergraph.
- Hypergraph partitioning results in identification of the blocks in a singly-bordered block diagonal matrix.
Instance p2756 with 10 blocks partitioning
1 Introduction to COIN
   - COIN-OR Foundation
   - Overview of Projects

2 Overview of Optimization Suite
   - Installing the COIN Optimization Suite
   - Documentation and Support

3 Entry Points
   - Modeling Systems
   - Python Tools
   - Command-line Tools
   - Building Applications

4 Advanced Development
   - SYMPHONY
   - DIP
   - CHiPPS
   - Working with Source
• 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.

```cpp
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

\[
\begin{align*}
\text{min} & \quad c^T x \\
\text{s.t.} & \quad Ax \leq b \\
& \quad x_i \in \mathbb{Z} \quad \forall \ i \in I
\end{align*}
\]

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.
Build Tools

- Build system is based on the GNU auto tools.
  - 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.
Monolithic Builds from Source (*Nix)

- Suppose you want 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/1.6 CoinAll-1.6
cd CoinAll-1.6
mkdir build
  cd build
  ../configure --enable-gnu-packages -C --prefix=/path/to/install/location
  make -j 2
  make test
  make install
```

- Note that after building, the examples will be installed with Makefiles in project subdirectories.
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 and a build harness for build them.

We also produce libraries and pkg-config files.

- AMPL Solver Library
- Blas
- Lapack
- Glpk
- Metis
- MUMPS
- Soplex
- SCIP
- HSL
- 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`
- 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

- Over-riding variables: `CC`, `CXX`, `F77`
- `-prefix`
- `-enable-debug`
- `-enable-gnu-packages`
- `-C`
Building Individual Projects from Source (*Nix)

- Assuming some libraries are already installed in
  /path/to/install/location

**Tweaking a Single Library**

```
svn co http://projects.coin-or.org/svn/Cbc/stable/2.6/Cbc Cbc-2.6
cd Cbc-2.6
mkdir build
cd build
./configure --enable-gnu-packages -C --prefix=/path/to/install/location
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.
Building Individual Projects from Source (Windows)

- Building with either CYGWIN or MinGW compilers is just as on other *nix systems.
- For Visual Studio, it is possible to build with the \texttt{cl} compiler using the autotools!
- To build through the IDE, MSVC++ project files provided for most projects.
- Current standard version of the compiler is v10.
- Projects requiring Fortran are a problem with the MSVC++ IDE.

Keeping settings synced across all projects has always been painful.

\textbf{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.
COIN needs your help!

- Contribute a project
- Help develop an existing project
- Use projects and report bugs
- Volunteer to review new projects
- Develop documentation
- Develop Web site
- Chair a committee

Questions? & Thank You!