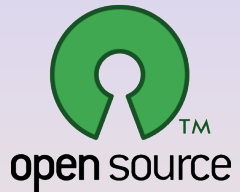


Accessible Analytics: Openness and Operations Research

Ted Ralphs



South African Operations Research Society Annual Meeting
Parys, South Africa, 15 September 2014

Outline

- 1 Accessibility and Openness
- 2 The Democratization of Computing
- 3 Open Movements Today
- 4 Introduction to COIN-OR
 - COIN-OR Foundation
 - Overview of Projects
- 5 The COIN-OR Optimization Suite
 - Modular Structure
 - Basic Building Blocks
- 6 Case Studies
- 7 Conclusions

Defining Access

What is it that we want to access? And why? What does accessibility mean?

- For **physical devices** or **technologies**, access usually means either
 - physical possession of the device or object,
 - the ability to look "inside" the box at a device's parts, or
 - access to the device over a network (e.g., computer)
- Access to **information**, on the other hand, connotes *understanding*, e.g.,
 - a clear explanation of the inner workings of some given technology (device, software), or
 - access to and an explanation of data describing a given phenomena.



Why is it Important?

- **Societal progress** requires sharing of and access to the inventions and ideas of others.
- Without access, the proverbial wheel will be repeatedly re-invented.
- Good science also requires access.
 - It is important to understand the relationship of your work to that of others.
 - The scientific method requires us to understand and replicate the ideas and experiments of others.



Access to technology can improve the lives of individuals.

What Limits Accessibility?

- For physical products, the end product may be difficult/expensive to replicate.
- Connectivity may prevent access to data and/or information, as well as remote access to devices.
- Difficulty in enabling transfer of ideas
 - Prohibition on disclosure due to commercial interests.
 - Closed/expensive publication venues.
 - Too much information (“noise”).
 - Technically “deep” ideas that cannot be easily understood.
 - Lack of availability of implementations or specific designs.



It takes effort on the part of individuals to make technology accessible and there are few rewards.

What Is “Openness”?

- Accessibility is closely related to another important concept: openness.
- The **Open Knowledge Definition** describes well what is required for knowledge to be "open".
 - Free access must be granted.
 - Free redistribution must be allowed.
 - Modifications and derivative works must be allowed.
 - Technological restrictions must not be imposed.
 - Respect must be given to attribution requests.
 - Respect must be given to integrity of the original work
 - No persons or groups may be discriminated against.
 - No discrimination can be made against fields of endeavor
 - The license cannot be restricted to a “package.”
 - There can be no restriction on the distribution of combined works.



Open Technologies

Over time, technologies have tended to evolve towards a more“open” development model through a number of different mechanisms.

- The model may be pursued as a matter of philosophy.
- Some closed technologies developed by governments have later been placed in the public domain (GPS, Internet, ...).
- Commerical technologies may be reverse-engineered or re-invented by hobbyists and academics.
- Technologies may be opened by the commerical entity that developed them either
 - to encourage wide adoption,
 - because they have been “commoditized” and are no longer profitable, or
 - to ensure control of a standard.



Openness and Accessibility in Analytics

- The more open we can be, the faster we will make progress as a field.
- More importantly, openness helps "newcomers" to better understand what analytics are about.
- This leads to more users and eventually more developers of our technologies.
- The phrase "accessible analytics" indicates a healthy spirit of inclusiveness and outreach.
- Later in the talk, I'll touch on Eric Raymond's notion of the "cathedral" versus the "bazaar" as a model for development.
- Although the quiet and orderly nature of the cathedral may be enticing, the bazaar is where business gets done!

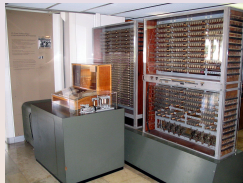
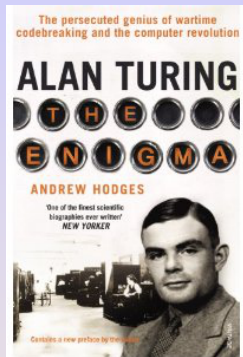
Openness is the means to the end of accessibility

Outline

- 1 Accessibility and Openness
- 2 The Democratization of Computing**
- 3 Open Movements Today
- 4 Introduction to COIN-OR
 - COIN-OR Foundation
 - Overview of Projects
- 5 The COIN-OR Optimization Suite
 - Modular Structure
 - Basic Building Blocks
- 6 Case Studies
- 7 Conclusions

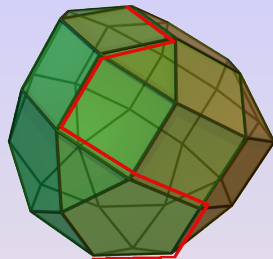
Foundations (1930-1946)

- The foundations for modern computing were laid out by two pioneers.
 - **Charles Babbage** is credited with designing the first computing machine.
 - **Alan Turing** developed the core theoretical and practical notions on which computer science is still based.
- Early computers
 - **Z3**: first electromechanical, programmable, fully automatic, digital computer.
 - **Colossus**: Turing built this first electronic digital programmable computer (1944)
 - **ENIAC**: first Turing-complete digital computer (1946).



The Simplex Algorithm and the Transistor (1947)

- In 1947, two important and seemingly unrelated discoveries were made.
 - George Dantzig invented the Simplex method.
 - Bardeen, Brattain, and Shockley demonstrated the first transistor.
- The evolution of hardware technology and mathematical programming have since been inexorably linked.
- Each step forward in computing technology is immediately exploited by researchers in mathematical programming.



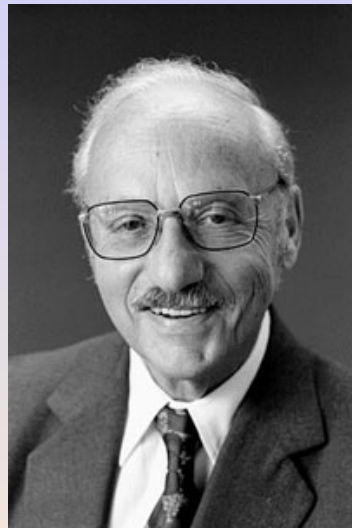
Dawn of Commerical Computing (1947-1960)

- Commercial computers were first introduced in 1951 (Ferranti Mark 1, LEO I, UNIVAC I).
- IBM introduced its first fully electronic data processing system (701) and the more versatile and popular IBM 650 in 1954.
- Initially, IBM bundled “free” software with its computers.
 - It could be redistributed.
 - Source code was included and could be modified.
 - The software was not seen as a separate product.
- In the 1960s, IBM “unbundled” its software and this led to an era dominated by proprietary software.



Early Linear Programming Codes (1947-1960)

- Until 1952, most LP computation was done on punched card calculators (which accounts for many of the eccentricities of the MPS format!)
- The first general-purpose implementation of the simplex algorithm is reported to have been on the SEAC computer at the National Institute of Standards around 1952.
- Dantzig and Orchard-Hays laid the groundwork for modern implementations of the simplex algorithm at RAND between 1952-1956.
- They implemented the simplex algorithm on an IBM 701 in 1954-55.
- This was followed by an improved implementation on a 704 in 1955-56.



Unix and the Dawn of Open Standards (1970-1984)

- Much of the history of open source is tied to the history of Unix
- Early versions of Unix were developed by a team at AT&T Bell Labs headed by Ken Thompson and Dennis Ritchie based on earlier system called *Multics*.
- In the late 70s and early 80s, two separate movements evolved.



- On the U.S. West coast, the Computer Science Research Group at UC Berkeley was developing BSD Unix based on AT&T's Unix implementation (1978).
- On the U.S. East coast, Richard Stallman founded the Free Software Foundation and began to develop the free GNU (GNU is Not Unix) operating system (1983).
- The concept of “open systems” was developed by vendors concerned about control of system interfaces (1984).

Linux, Free BSD, and the Evolution of Sharing (1985-1995)

- In the late 1980s and early 1990s, the rise of USENET and the Internet lead to the sharing of code and the development of strong user communities.
- The X Windows System was developed and was the first example of an open source platform developed by a consortium of companies.
- In 1991-1992, several important developments were set to change the landscape altogether.



Linux



FreeBSD

- The 386BSD became the first completely free operating system, including a kernel and utilities, mostly under the BSD open source license.
- Linus Torvalds began implementing the first version of the Linux kernel, which was later combined with GNU utilities to make the free GNU/Linux operating system.

Free Software Movement (1983-)

Richard Stallman was the first to formalize the notion of "free" software when he developed the **Free Software Definition**:

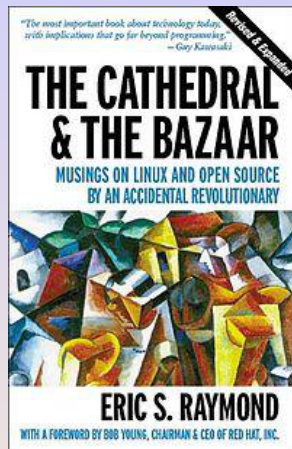
- The freedom to run the program, for any purpose.
- The freedom to study how the program works, and change it so it does your computing as you wish.
- The freedom to redistribute copies so you can help your neighbor.
- The freedom to distribute copies of your modified versions to others.



The GNU Public License codified these freedoms and added the notion of "copyleft", which essentially required linked programs to also be under the GPL.

Open Source Movement (1997-)

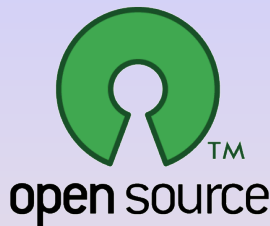
- In the late 1990s, a rival movement developed in response to what some saw as the extremism of the Free Software Foundation.
- There was a need to establish a more "industry-friendly" notion of "free" software.
- A catalyst for the movement was the publication of Eric Raymond's *Cathedral and the Bazaar*.
- This soon led to the founding of the Open Source Initiative.
- The Open Source Definition codified the foundations beliefs about what openness really entails.



Open Source Definition

Requirements of the Open Source Definition:

- Free redistribution must be allowed.
- Source code must be included.
- Modifications and derived works must be allowed.
- Respect for integrity of the author's source code must be respected (as a compromise).
- Persons or groups cannot be discriminated against.
- No fields of endeavor, like commercial use, can be discriminated against.
- The license needs to apply to all to whom the program is redistributed.
- The license must not be specific to a product.
- The license must not contaminate other software.
- The license must be technology-neutral.

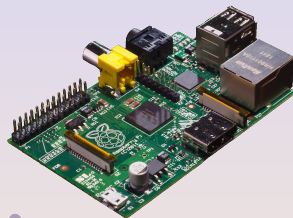


Outline

- 1 Accessibility and Openness
- 2 The Democratization of Computing
- 3 Open Movements Today**
- 4 Introduction to COIN-OR
 - COIN-OR Foundation
 - Overview of Projects
- 5 The COIN-OR Optimization Suite
 - Modular Structure
 - Basic Building Blocks
- 6 Case Studies
- 7 Conclusions

Open Hardware

- There is an Open Source Hardware Definition similar to the Open Source Definition.
- It requires hardware be released with documentation and design files, in addition to required software.



Open Publishing

- Open publishing is a bit more of a mixed bag.
- On the one hand, the combination of LaTeX and the Internet make it simple to self-publish.
- On the other hand, although more open access journals are arising, the vast majority of journals are still traditional.
- The tenure and promotion process in the U.S. and elsewhere have made it difficult to break the strangle-hold of traditional publishers.



Open Software



Open Operations Research

- The field of operations research faces all of the above frictions associated with the sharing of ideas.
- Prior to 2000, operations research had few “open” venues for the distribution of ideas.
 - Papers were most often not available open access.
 - Source code required to replicate experiments was often not available.
 - Progress in computation was relatively slow.



This was the impetus for the development of COIN-OR

Outline

- 1 Accessibility and Openness
- 2 The Democratization of Computing
- 3 Open Movements Today
- 4 Introduction to COIN-OR**
 - COIN-OR Foundation
 - Overview of Projects
- 5 The COIN-OR Optimization Suite
 - Modular Structure
 - Basic Building Blocks
- 6 Case Studies
- 7 Conclusions

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.

The COIN Boards

The COIN-OR Foundation is governed by two boards.

Strategic Leadership Board


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

Technical Leadership Council

- Miles Lubin
- Kipp Martin
- Ted Ralphs (Chair)
- Haroldo Santos
- John Siirola
- 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?



You received a payment

May 2, 2013 08:02:42 PDT
Transaction ID: [8007-794236-00002](#)

Hello COIN-OR Foundation, Inc,

You received a payment from [XXXXXXXXXXXX](#) for Donation

Customer details

Customer name:	XXXX, C L S C O R F
Customer email:	XXXXXXXXXXXX@XXXX.XXXXXX.XXXXXX
Profile ID:	XXXX-XXXX-XXXX-XXXX
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

- 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...

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!

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
- **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
- **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

- 1 Accessibility and Openness
- 2 The Democratization of Computing
- 3 Open Movements Today
- 4 Introduction to COIN-OR
 - COIN-OR Foundation
 - Overview of Projects
- 5 The COIN-OR Optimization Suite**
 - **Modular Structure**
 - **Basic Building Blocks**
- 6 Case Studies
- 7 Conclusions

The COIN-OR Optimization Suite

- Many of the tools focused on solution of mathematical optimization models interopate.
- They are built 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 talk.

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

Building Blocks: Open Solver Interface

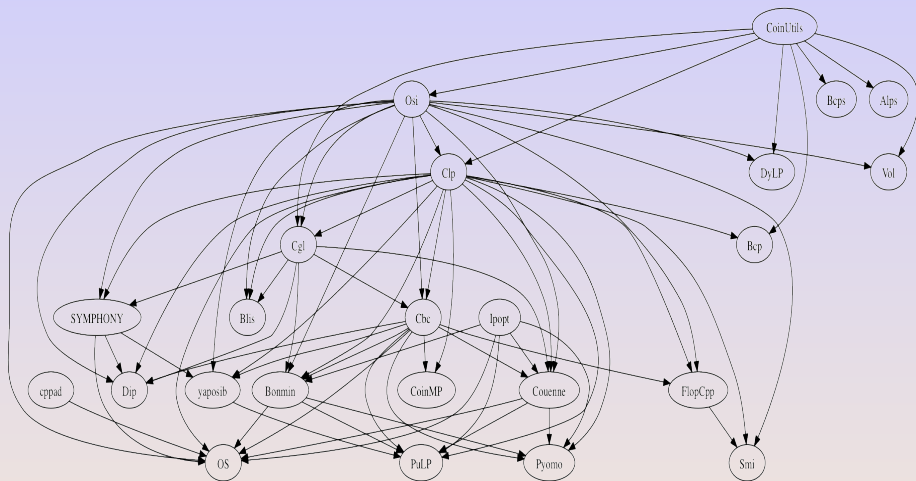
Uniform API for a variety of solvers:

- 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.
 - 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 possibility of providing Linuxbrew packages
- 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.
- Other ways of obtaining COIN include downloading it through a number of modeling language front-ends (more on this later).

Outline

- 1 Accessibility and Openness
- 2 The Democratization of Computing
- 3 Open Movements Today
- 4 Introduction to COIN-OR
 - COIN-OR Foundation
 - Overview of Projects
- 5 The COIN-OR Optimization Suite
 - Modular Structure
 - Basic Building Blocks
- 6 Case Studies
- 7 Conclusions

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

$$\begin{aligned} \min \quad & \sum_{j=1}^n c_j y_j + \sum_{i=1}^m \sum_{j=1}^n d_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^n x_{ij} = 1 && \forall i \\ & \sum_{i=1}^m w_{ij} x_{ij} \leq W_j && \forall j \\ & x_{ij} \leq y_j && \forall i, j \\ & x_{ij}, y_j \in \{0, 1\} && \forall i, j \end{aligned}$$

PuLP Basics: Facility Location Example

```
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,
```

DipPy: Modeling Decomposition (Mike O'Sullivan)

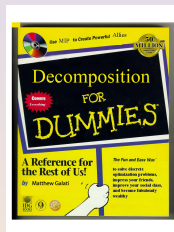
DIP Framework

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

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

Methods

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



⇐ *Joke!*

DipPy Basics: Facility Location Example

```
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.

Optimization Services (OS)

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

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

Using AMPL with OS

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

```
model hs71.mod;  
# tell AMPL that the solver is OSAmplClient  
option solver OSAmplClient;  
  
# now tell OSAmplClient to use Ipopt  
option OSAmplClient_options "solver ipopt";  
  
# now solve the problem  
solve;
```

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

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

OS: Solving a Problem on the Command Line

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

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

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

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

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

OS: Remote Solves

The OSSolverService can be invoked to make remote solve calls.

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

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

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

OS: Specifying a Solver

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

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

- `clp`
- `dylp`

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

- `cbc`
- `symphony`

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

- `ipopt`

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

- `bonmin`
- `couenne`

OS: File formats

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

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

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

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

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

Step 4: Build solver-specific model instance

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

Step 5: Solve the problem.

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

Building an OS Instance

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

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

Building an OS Instance (cont.)

- Create an `OSInstance` object.

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

- Put some variables in

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

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

Outline

- 1 Accessibility and Openness
- 2 The Democratization of Computing
- 3 Open Movements Today
- 4 Introduction to COIN-OR
 - COIN-OR Foundation
 - Overview of Projects
- 5 The COIN-OR Optimization Suite
 - Modular Structure
 - Basic Building Blocks
- 6 Case Studies
- 7 Conclusions

Conclusions

- COIN-OR is just one of many organizations moving the operations research community towards more openness and accessibility.
- It has been gratifying to see the growth and wide adoption of COIN-OR, but we need your help!
- Everyone can play a part.
- If you are passionate about openness, join us!.

Let's change the world!!