Multistage Discrete Optimization
Part IV: Software

Ted Ralphs[1]
Joint work with Suresh Bolusani[1], Scott DeNegre[3],
Menal Güzelsoy[2], Anahita Hassanzadeh[4], Sahar Tahernajad[1]

[1]COR@L Lab, Department of Industrial and Systems Engineering, Lehigh University
[2]SAS Institute, Advanced Analytics, Operations Research R & D [3]The Hospital for Special Surgery [4]Climate Corp

Friedrich-Alexander-Universität Erlangen-Nürnberg, 20-21 March 2017

Outline

Outline

Brief History of COIN-OR

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

What is COIN-OR Today?

The COIN-OR Foundation

- A non-profit foundation promoting the development and use of interoperable, open-source software for operations research.
- A consortium of researchers in both industry and academia dedicated to improving the state of computational research in OR.
- A venue for developing and maintaining standards.
- A forum for discussion and interaction between practitioners and researchers.

The COIN-OR Repository

- A collection of interoperable software tools for building optimization codes, as well as a few stand alone packages.
- A venue for peer review of OR software tools.
- A development platform for open source projects, including a wide range of project management tools.

The COIN Boards

The COIN-OR Foundation is governed by two boards.

Strategic Leadership Board

- Kevin Furman
- Bill Hart
- Alan King (Treasurer)
- Andrew Mason
- Giacomo Nannicini
- Ted Ralphs (TLC Rep)
- Matt Saltzman (President)

Technical Leadership Council

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

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

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

COIN-OR Projects Overview: Interfaces and Solver Links

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

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

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

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

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

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

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

COIN-OR Projects Overview: Frameworks

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

  Project Manager: Laci Ladanyi
- CHiPPS: A framework for developing parallel tree search algorithms.

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

  Project Manager: Ted Ralphs

COIN-OR Projects Overview: Automatic Differentiation

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

  Project Manager: Andrea Walther

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

  Project Manager: Brad Bell

COIN-OR Projects Overview: Graphs

- GiMPy and GrUMPy: Python packages for visualizing algorithms

  Project Manager: Ted Ralphs

- Cgc: Coin graph class utilities, etc.

  Project Manager: Phil Walton

- LEMON: Library of Efficient Models and Optimization in Networks

  Project Manager: Alpar Juttner

COIN-OR Projects Overview: Miscellaneous

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

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

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

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

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

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

  Project Manager: David Hawes

Outline

The SYMPHONY MILP Solver Framework

- Warm-starting is an inherently useful technique in the solution of these algorithms.
- Since one of our constraints involves the value function, we must (either implicitly or explicitly construct an approximation of this is function.
- This can be done in a number of ways, as we'll see.
- SYMPHONY is an MILP solver framework that supports
  - Warm starting of the solution process.
  - Sensitivity analysis
  - Exporting of the branch-and-bound dual function.

SYMPHONY

Overview of Features

- An open source, customizable, callable library for solving mixed integer linear programs. with a wide variety of customization options.
- Core solution methodology is branch and cut.
- Includes advanced features, such as solution of biobjective problems.
- Supports shared and distributed parallel solution modes.
- Extensive documentation available.
- Available for download at `projects.coin-or.org/SYMPHONY`.
- All of the methods discussed in this talk are in SYMPHONY 5.6.

Available Customized Solvers Built with SYMPHONY

- Generic MILP
- Multicriteria MILP
- Traveling Salesman Problem
- Vehicle Routing Problem
- Mixed Postman Problem
- Set Partitioning Problem
- Matching Problem
- Network Routing

SYMPHONY: Support for Warm Starting

Currently supported

- Change to objective function (no reduced cost fixing during generation of warm start).
- Change to right hand side (cuts are discarded when resolving).
- Changes to variable bounds.
- Addition of columns.

Coming soon

- Addition of constraints (easy).
- Changes to right hand side without discarding cuts (not so easy).
- Changes to objective function with reduced cost fixing (not so easy).

Basic Sensitivity Analysis

SYMPHONY will calculate bounds after changing the objective or right-hand side vectors.

```
int main(int argc, char **argv)
{
   OsiSymSolverInterface si;
   si.parseCommandLine(argc, argv);
   si.loadProblem();
   si.setSymParam(OsiSymSensitivityAnalysis,
                  true);
   si.initialSolve();
   int ind[2];
   double val[2];
   ind[0] = 4;   val[0] = 7000;
   ind[1] = 7;   val[1] = 6000;
   lb = si.getLbForNewRhs(2, ind, val);
}
```

Warm Starting Example (Parameter Modification)

The following example shows a simple use of warm starting to create a dynamic algorithm.

```
int main(int argc, char **argv)
{
   OsiSymSolverInterface si;
   si.parseCommandLine(argc, argv);
   si.loadProblem();
   si.setSymParam(OsiSymFindFirstFeasible,true);
   si.setSymParam(OsiSymSearchStrategy,
                  DEPTH_FIRST_SEARCH);
   si.initialSolve();
   si.setSymParam(OsiSymFindFirstFeasible,false);
   si.setSymParam(OsiSymSearchStrategy,
                  BEST_FIRST_SEARCH);
   si.resolve();
}
```

Warm Starting Example (Problem Modification)

This example shows how to warm start after problem modification.

```
int main(int argc, char **argv)
{
   OsiSymSolverInterface si;
   CoinWarmStart ws;
   si.parseCommandLine(argc, argv);
   si.loadProblem();
   si.setSymParam(OsiSymNodeLimit, 100);
   si.initialSolve();
   ws = si.getWarmStart();
   si.resolve();
   si.setObjCoeff(0, 1);
   si.setObjCoeff(200, 150);
   si.setWarmStart(ws);
   si.resolve();
}
```

Improving Variable Bounds

- Prior to any warm solve, we can improve variable bounds by computing generalized reduced costs.
- For each variable, we increase its lower bound temporarily and perform a sensitivity analysis.
- If such a bound change leads to a dual (lower) bound (on the modified instance) exceeding the known primal (upper) bound (on the instance), we can improve the variable's upper bound.
- The same can be done to improve lower bounds.

Outline

Implementation

The Mixed Integer Bilevel Solver (MibS) implements the branch and bound framework and heuristic methods described here using software available from the Computational Infrastructure for Operations Research (COIN-OR) repository.

COIN-OR Components Used

- The COIN High Performance Parallel Search (CHiPPS) framework to perform the branch and bound.
- The COIN Branch and Cut (CBC) framework for solving the MILPs.
- The COIN LP Solver (CLP) framework for solving the LPs arising in the branch and cut.
- The Cut Generation Library (CGL) for generating cutting planes within CBC.
- The Open Solver Interface (OSI) for interfacing with CBC and CLP.

Please visit `www.coin-or.org` for information on obtaining these codes.

Primary MibS Classes

The primary classes the comprise MibS are as follows:

MibS Classes

1. `MibSModel`: Derived from the virtual BLIS class `BlisModel` and stores information about the original problem.
2. `MibSCutGenerator`: Derived from the virtual BLIS class `BlisConGenerator`, and used to generate cuts when CHiPPs finds integer, bilevel infeasible solutions.
3. `MibSSolution`: Derived from the virtual BLIS class `BlisSolution` and stores and prints integer bilevel feasible solutions.
4. `MibSBilevel`: Specific to MibS and used to test bilevel feasibility of integer solutions.
5. `MibSHeuristic`: Specific to MibS and used to generate heuristic solutions.

Main Parameters

- Branching
  - `branchStrategy`: `linking, fractional`
  - b̂lisBranchStrategy: `strong, pseudocost, reliability`
- Cutting
  - Turn all classes of cuts on and off (automatic defaults)
  - Determine how often to generate cuts
- Primal Heuristics
  - Turn all heuristics on and off (automatic defaults).
  - Determine how often to execute heuristics.
- `useLinkingSolutionPool`: `true, false`.
- When to solve (SS-MILP) and (UB).

Getting and Installing MibS

MibS source is available at:

https://github.com/tkralphs/MibS

Basic install on Linux/OS X

```
git clone https://github.com/tkralphs/MibS
cd MibS
git clone https://github.com/coin-or-tools/BuildTools/
BuildTools/get.dependencies fetch
BuildTools/get.dependencies build --parallel-jobs=2
BuildTools/get.dependencies install --prefix=/path/to/inst/dir
```

File Format

- MibS currently supports specifying main problem data in MPS format.
- An auxiliary file contains information indicating which variables and constraints are associated with which level.
  - `M`: Number of lower level variables
  - `N`: Number of lower level constraints
  - `LC`: Index of a lower-level variable
  - `LR`: Index of a lower-level constraint
  - `LO`: Coefficients of lower-level objective
  - `OS`: Lower-level objective sense

-
  ```
  ./mibs -Alps_instance file.mps -MibS_auxiliaryInfoFile
  aux_file.txt
  ```

Computational Results

- All computations were done on a Linux (Debian 6.0.9) box 16 AMD 800 MHz processors and 32 GB RAM.
- Test set was as follows.

Table 1: The summary of data sets

Data Set	First-level Vars Num	Second-level Vars Num	First-level Cons Num	Second-level Cons Num	First-level Vars Type	Second-level Vars Type	Size
INTER-DEN	10	10	1	11	binary	binary	20
IBLP-DEN	5-15	5-15	0	20	discrete	discrete	50
IBLP-FIS	4-2481	2-2480	0	16-4944	binary	binary	24
MIBLP-XU	10-460	10-460	4-184	4-184	discrete	continuous, discrete	100

Solving Problems (SS-MILP) and (UB)

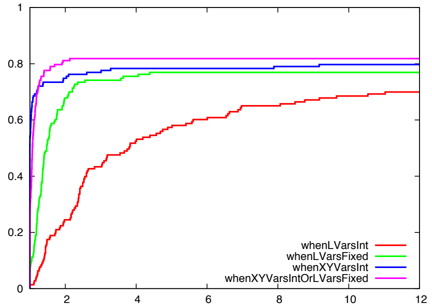
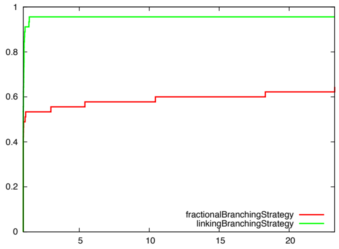
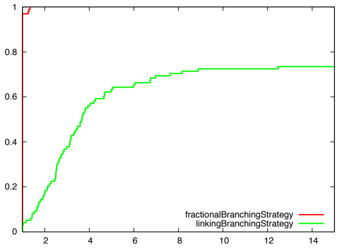


Figure 1: Impact of the parameters for solving problems (SS-MILP) and (UB)

Impact of Branching Strategy



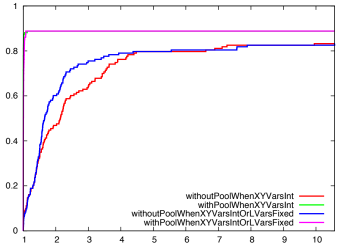
(a) $r_1 \leq r_2$



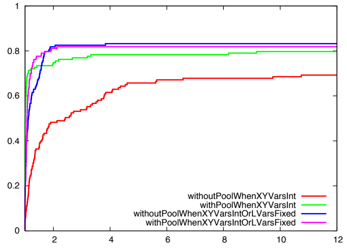
(b) $r_1 > r_2$

Figure 2: Impact of the branchStrategy parameter.

Impact of Linking Solution Pool



(a) $\overset{\frown}{\text{f}}$ractional branching strategy



(b) $\overset{\frown}{\text{l}}$inking branching strategy

Figure 3: Impact of the linking solution pool.

Outline

Generalized Bender Implementation

- We've implemented the generalized Benders Algorithm from the previous slides.
- We use SYMPHONY as the solver for the second-stage because it is capable of exporting the required dual functions.
- Either CPLEX or Cbc can be used to solve the master problem.
- The source will be made available open source "soon."
- I first have to put my acronym generation skills to the test :).

SSLP Instances

We apply the algorithm to the SSLP test instances from SIPLIB. The instances have the following properties. The last column shows the deterministic equivalent solution time in seconds.

	DEP			2nd Stage				
Instance	cons	bin	int	cons	bin	int	Time (s)	% Gap
sslp-5-25(25)	751	3130	125	30	130	5	3.17	0
sslp-5-25(50)	1501	6255	250	30	130	5	4.22	0
sslp-5-25(100)	3001	12505	500	30	130	5	14.34	0
sslp-10-50(50)	3001	25010	500	60	510	10	3600+	70
sslp-10-50(100)	6001	50010	1000	60	510	10	3600+	15
sslp-15-45(5)	301	3390	75	60	690	15	3600+	1
sslp-15-45(10)	601	6765	150	60	690	15	1088.69	0

Table 2: The deterministic equivalent of SSLP instances

where "DEP" and "2nd Stage" correspond to the deterministic equivalent and the second stage problems and "cons", "bins" and "int" respectively represent the number of constraints, binary variables and general integer variables in the corresponding problem.

SSLP Instances

Instance	Iteration	Size	Time (s)	%Gap
sslp-5-25(25)	16	(2493, 3639)	182.58	0
sslp-5-25(50)	18	(789, 1821)	12.16	0
sslp-5-25(100)	18	(1322, 3410)	27.91	0
sslp-10-50(50)	8	(40K, 71K)	-	23
sslp-10-50(100)	8	(74K, 125K)	-	9
sslp-15-45(5)	7	(29K, 56K)	-	99
sslp-15-45(10)	26	(17K, 29K)	-	52

Table 3: Generalized Benders' algorithm applied to SSLP instances

- The results are generated using the MILP solver SYMPHONY version WS revision 2522 and CPLEX 12.5.
- The tests were performed on a 16-core Linux box with 800 MHz AMD processors and 31 GB RAM compiled with g++.
- SSLP 10 and 15 runs were in parallel using 16 cores, other runs sequential.
- The "Gap%" column refers to the relative gap between the upper bound and lower bound of the Generalized Benders' algorithm.

Analysis

- Getting this all to work well is a PITA!

Figure 4: Jack-in-the-Box Chicken Fajita Pita

- It appears possible to generalize the master problem to work for more general bilevel problems, but there are some potential stumbling blocks.

Outline

Conclusions

- The theory underlying these algorithms is maturing.
- Many of the computational tools necessary for experimentation now also exist.
- Computationally, we have looked primarily at the recourse (stochastic programming) and interdiction (zero sum) cases.
- Substantial progress has been made on these, but they appear much easier than the general case.
- In future work, we plan to leverage what we've learned to develop methodology for the general case.

Shameless Promotion: Free Stuff!

- CHiPPS: Parallel tree search framework
- DIP/DipPy: Decomposition-based modeling language and MILP solver
- DiSCO, OsiConic, CglConic: Mixed integer conic solver
- MibS: Mixed integer bilevel solver
- SYMPHONY: MILP solver framework with bicriteria, warm starting, etc.
- GiMPy, GrUMPy: Visualizations and illustrative implementations for graph and optimization algorithms.
- CuPPy: Cutting planes in Python
- Value Function: Algorithm for constructing value functions
- And more...

`http://github.com/tkralphs`