The COIN-OR Open Solver Interface: Technology Overview

Matthew Saltzman
Mathematical Sciences
Clemson University

László Ladányi
T. J. Watson Research Center
IBM

Ted Ralphs
Industrial and Systems Engineering
Lehigh University

CORS/INFORMS Banff
May 2004

Outline

- The COIN-OR Project
- COIN-OR Components
- The Current OSI API
- The Next-Generation OSI API
- Conclusion

What is COIN-OR?

The COIN-OR Project

- A consortium of researchers in industry and academia dedicated to improving the state of computational research in OR.
- An initiative promoting the development and use of interoperable, open-source software for operations research.
- Now a non-profit corporation: the COIN-OR Foundation, Inc.

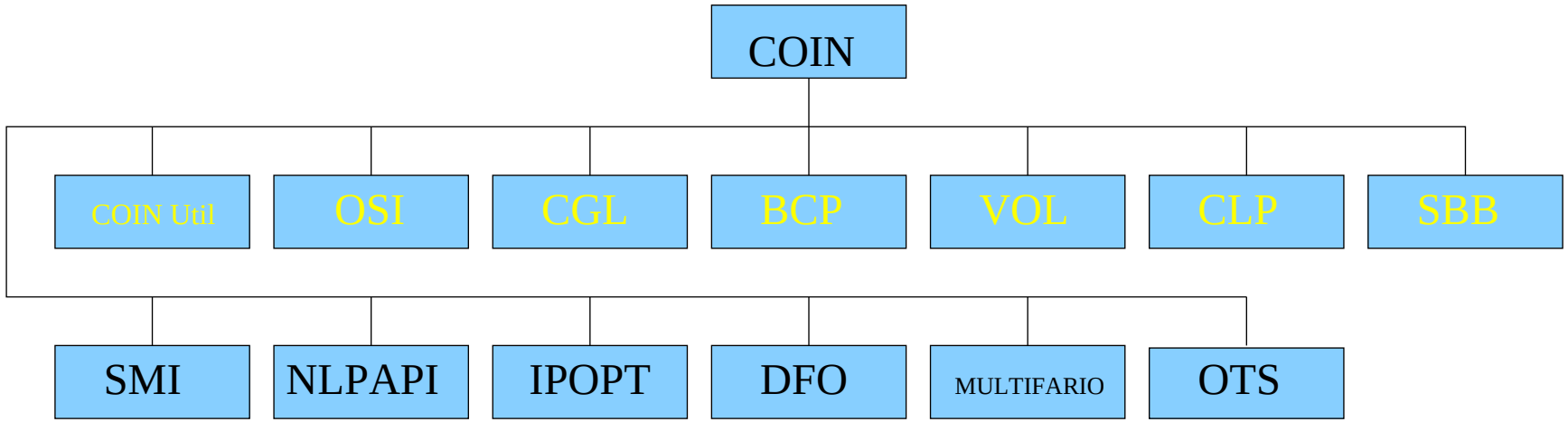
The COIN-OR Repository

- A library of interoperable software tools for building optimization codes, plus standalone packages.
- A venue for peer review of OR software tools.
- A development platform for open source projects, including a CVS repository and other tools.

Our Agenda

- Accelerate the pace of research in computational OR.
  - Reuse instead of reinvent.
  - Reduce development time and increase robustness.
  - Increase interoperability.
- Provide for software what the open literature provides for theory.
  - Peer review of software.
  - Free distribution of ideas.
  - Promotion of principles of good scientific research.
- Define standards and interfaces that allow software components to interoperate.
- Increase synergy between various development projects.
- Provide robust, open-source tools for practitioners.

Components of the COIN-OR Library



- Branch-cut-price toolbox
  - COIN Utilities
  - OSI: Open Solver Interface
  - CGL: Cut Generator Library
  - BCP: Branch-Cut-Price Framework
  - VOL: Volume Algorithm
  - CLP: COIN-OR LP Solver
  - SBB: Simple Branch & Bound
- Other components
  - SMI: Stochastic Modeling Interface
  - NLPAPI: Nonlinear Solver Interface
  - IPOPT: Interior Point Optimization (Nonlinear)
  - DFO: Derivative Free Optimization
  - MULTIFARIO: Solution Manifolds
  - OTS: Open Tabu Search

The Open Solver Interface Component

Purpose:

- A single API providing access to a variety of embedded solver libraries.
- Originally conceived as a "sandbox" for research on cutting planes, etc.

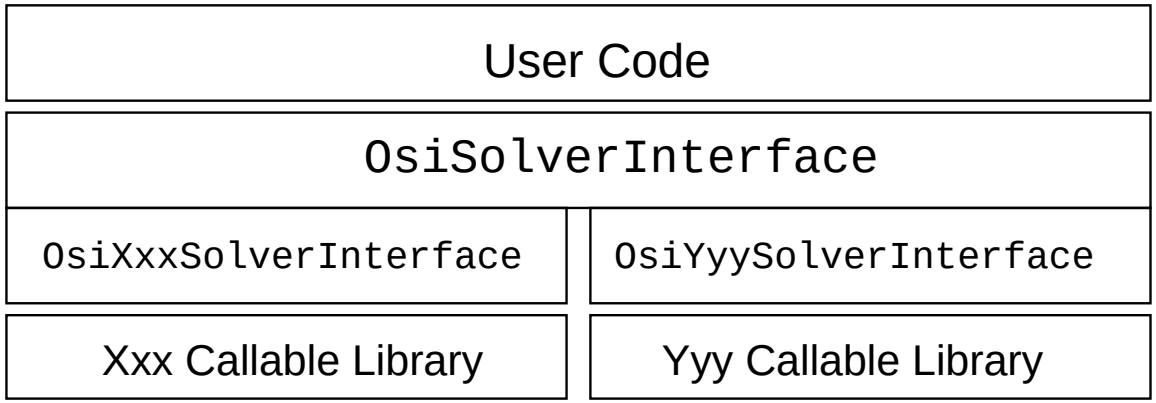
Current version features:

- Create/modify LP/MIP model loaded in solver.
- Access basic features of LP solvers.
- Modify model (add cutting plane inequalities generated via CGL) and resolve.
- Add-on provides access to simplex-specific features (only supported for some solvers).
- Can call MIP solver.

Supported Solvers

- COIN-LP (COIN-OR LP Solver, open source)
- CPLEX (ILOG, commercial)
- dylp (BonsaiG LP Solver, open source)
- FortMP (OptiRisk Systems, commercial)
- GLPK (GNU LP Kit, open source)
- Mosek (Mosek ApS, commercial, under construction)
- OSL (IBM, commercial)
- SoPlex (Konrad-Zuse-Zentrum für Informationstechnik Berlin, free for academic use)
- Volume (COIN-OR, open source)
- XPRESS (Dash Optimization, commercial)
- Add yours here. . .

Basic Design (Current)



- C++ classes
  - OsiSolverInterface base class
  - OsiXxxSolverInterface derived class for solver Xxx
- User writes code once using OSI API.
- Code works "out of the box" with any supported solver.
- Solver is instantiated as part of declaration of problem object.
  - Can be hidden from most of user code through object cloning.
  - Change solvers by recompiling `main()`, relinking.

Example `main()`

```
#if defined(COIN_USE_CPX)
#include "OsiCpxSolverInterface.hpp"

typedef OsiCpxSolverInterface
          RealSolverInterface;
#elif defined(COIN_USE_OSL)
#include "OsiOslSolverInterface.hpp"

typedef OsiOslSolverInterface
          RealSolverInterface;
#elif defined(COIN_USE_XPR)
#include "OsiXprSolverInterface.hpp"

typedef OsiXprSolverInterface
          RealSolverInterface;
#else
#error "Must define a solver."
#endif
```

```
void solve(
    const OsiSolverInterface *emptySi,
    const char *mpsfile,
    const double minmax);

int main(int argc, const char *argv[])
{
  // Arg 1: filename
  // Arg 2: "min" or "max"
  // Set minmax = 1.0 for min, -1.0 for max.

  // Instantiate solver interface
  RealSolverInterface si;

  solve(&si, filename, minmax);
  return 0;
}
```

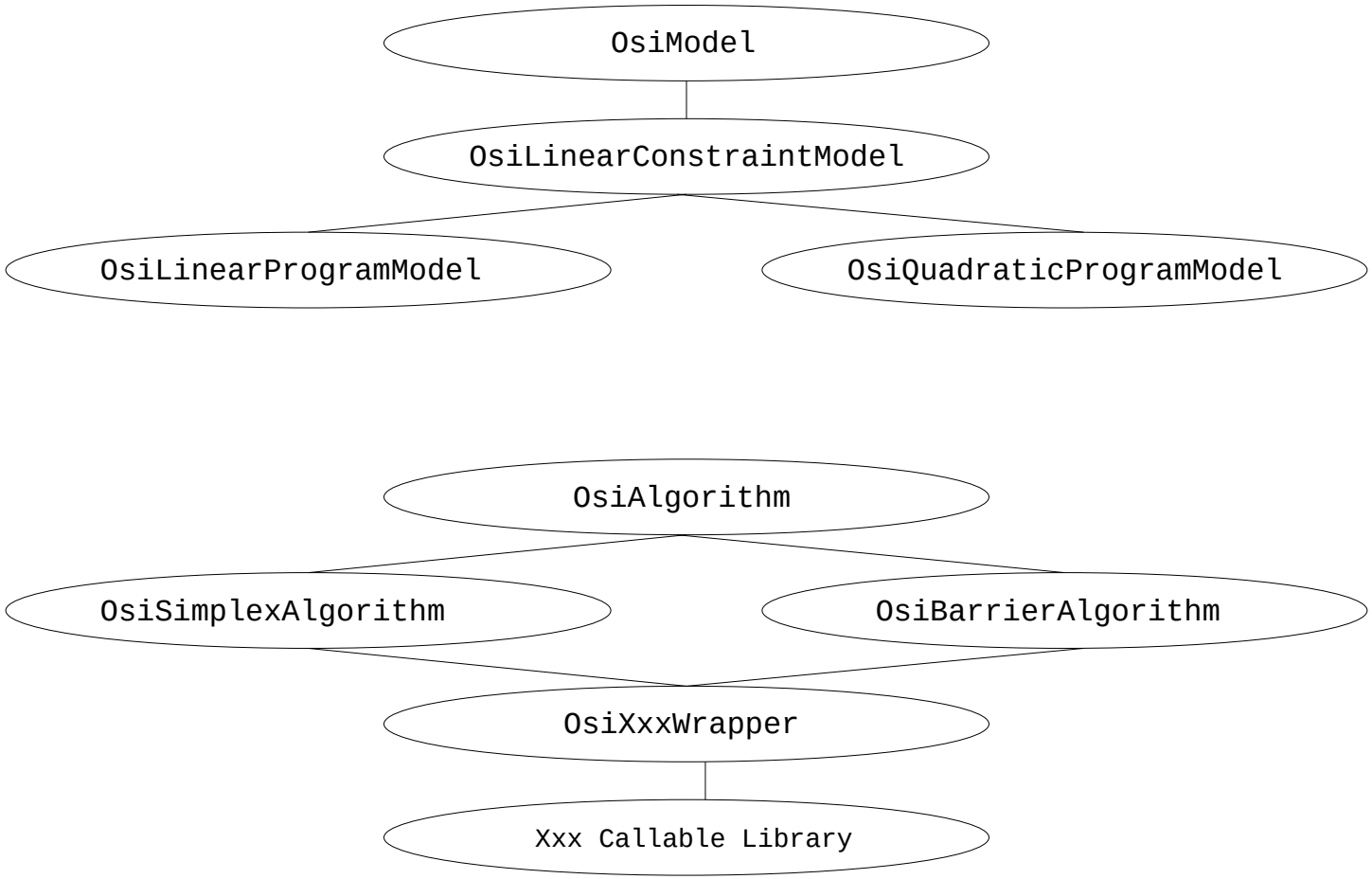
Example `solve()`

```
#include <iostream>
#include "OsiSolverInterface.hpp"

void solve(const OsiSolverInterface *emptySi,
           const char *mpsfile, const double minmax)
{
  // *si dynamically inherits derived class of *emptySi.
  OsiSolverInterface *si = emptySi->clone();

  si->readMps(fn, "mps");    // Read problem
  si->setObjSense(minmax);   // Set objective sense

  si->initialSolve();        // Solve continuous problem
  std::cout << "LP rel value: " << si->getObjValue() << std::endl;

  // Iteratively add cuts and resolve...

  si->branchAndBound();       // Solve MIP Problem
  std::cout << "Obj fn value: " << si->getObjValue() << std::endl;

  const double * soln = si->getColSolution();
  for ( int i = 0;  i < si->getNumCols();  i++ )
    std::cout << "x[" << i << "] = " << soln[i] << std::endl;
}
```

Next Generation: Design Goals

- Solver independence.
  - Consistent problem representation across solvers.
  - Consistent algorithm behavior across solvers (as far as possible).
- Thin wrapper layer.
- Portable (ANSI/ISO C++), standard, open.
- Usable.
- Flexible, extensible.
  - Separate `OsiModel` and `OsiAlgorithm` base classes.
  - Multiple derived model classes.
  - Multiple derived algorithm classes. Parameters, status information, solutions, etc., appropriate for algorithm.
  - A model is not bound to a solver until an algorithm is invoked.

Example Class Hierarchy



Consistent Problem Representation/Behavior

- "Solver independent" vs. "solver agnostic"
  - Base class can create/view/modify a complete problem/solution representation.
- User sees consistent view of problem no matter which solver is used.
  - MPS file represents the same problem no matter which solver is used.
- User has some options (e.g., row bounds vs. rhs type-value-range) not tied to solver.
- Solver internal memory management is transparent to the user.

`OsiModel` Design Features

- An `OsiModel` is a collection of `OsiVariable` and `OsiConstraint` objects.
- Each such object has an associated `OsiDomain`—a type (e.g., continuous, integer, semi-continuous, etc.) and an interval. An `OsiConstraint` also defines a function that produces its value.
- `OsiConstraints` may serve as constraints or objectives.
- Derived classes can implement specially-structured constraints and objectives.
  - e.g., `OsiLinearConstraintModel` implements constraints using `CoinSparseMatrix`.
- Objects of types derived from `OsiModel` can be manipulated without being bound to an underlying solver.

Memory Management

Goal: Minimize performance penalty for extra layer.

- Passing a model to an algorithm must copy or transfer the model to the solver's internal representation (copy in/out vs. pointer-based).
- Once loaded, changes to the model must be reflected transparently to the solver's internal representation.
- The wrapper layer must do the minimum work necessary to maintain synchronization. Any caching of model data outside the solver's internal representation must be handled in the base class.
- Cache should be controllable by user (speed/size tradeoff).

Cache Management

- Wrapper class
  - **Get:**
    Query base class for cached copy.
    If cached copy does not exist, query solver and cache result.
    Return pointer to cached result.
  - **Set:**
    Update solver copy.
    Pass mods through to model or solver base class.
- Base class
  - **Get:**
    Return cached copy.
  - **Set:**
    Update or delete cached copy in model object..

Complete Algorithm Feature Set

- Solver Interface should (as much as possible) support features of underlying callable library.
- At least core features required for efficient use of embedded solver (algorithm-specific). E.g., for simplex codes:
  - Warm starts with user-defined basis.
  - Pivot-level control.
  - Equation solving with basis matrix (FTRAN/BTRAN).
- Solution/warm-start classes are algorithm class dependent.
- Embedded solver representation can be accessed for direct calls to solver API (solver-dependent, not portable).

Messaging

- Should not be tied to a particular interface (e.g., stdout/stderr vs. GUI dialog boxes).
- Should not be tied to a particular language (i18n).

Currently:

- CoinMessageHandler class provides rudimentary message channel interface.
- Complex message handlers can be created as derived classes.
- Each component provides XxxMessage class containing list of index-message pairs.

Solver Parameters

Goal: Unified parameter handling mechanism across all solvers.

Design:

- Common list of parameters and values for OSI layer.
- Database mapping OSI parameters to solver parameters and types (supplied in `OsiXxxWrapper`).
- Solvers should degrade gracefully if they don't understand "hints".

Conclusion

Build a better solver interface. . .

Conclusion

Build a better solver interface. . .

. . . and the world will beat a path to your CVS repository.