The COIN-OR Open Solver Interface: Design Issues for a Portable Solver API

Matthew Saltzman
Mathematical Sciences
Clemson University

INFORMS San Jose
November 17, 2002

Outline

- The COIN-OR Project
- COIN-OR Components
- The Open Solver Interface
- Performance and coding issues
- Messaging
- Parameter setting
- 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.

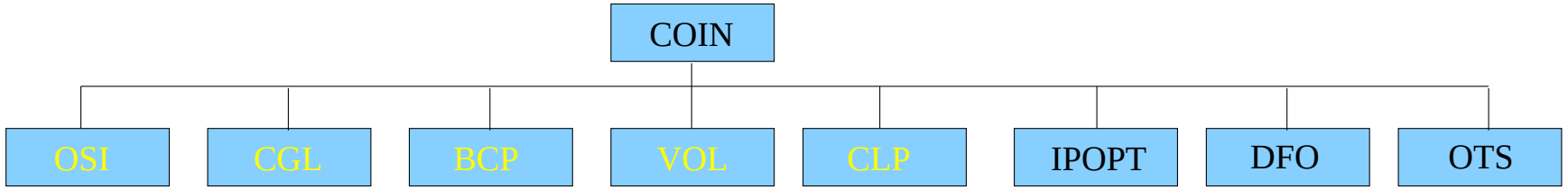
The COIN-OR Repository

- 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
  - OSI: Open Solver Interface
  - CGL: Cut Generator Library
  - BCP: Branch, Cut, and Price Framework
  - VOL: Volume Algorithm
  - CLP: COIN-OR LP Solver
- Stand-alone components
  - IPOPT: Interior Point Optimization (Nonlinear)
  - DFO: Derivative Free Optimization
  - OTS: Open Tabu Search

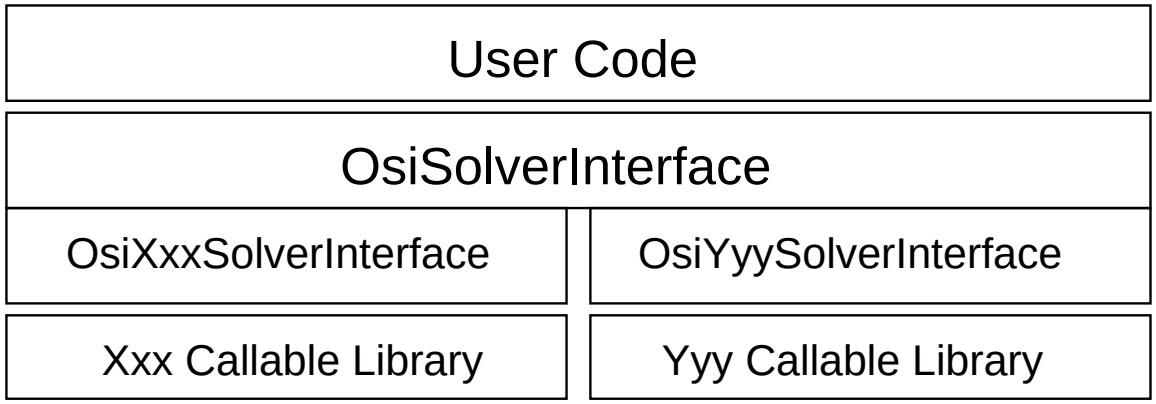
The Open Solver Interface Component

Purpose: A single API providing access to a variety of embedded solver libraries.

Solvers Supported:

- CLP (COIN-OR LP Solver, open source)
- CPLEX (ILOG, commercial)
- DyLP (BonsaiG LP Solver, open source)
- GLPK (GNU LP Kit, open source)
- 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 of OSI



- 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 *si0,
    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.h>
#include "OsiSolverInterface.hpp"

void solve(const OsiSolverInterface *emptySi,
           const char *mpsfile, const double minmax)
{
  // *si dynamically inherits derived class of *si0
  OsiSolverInterface *si = emptySi->clone();

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

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

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

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

Design Goals

- Consistent problem representation.
- Consistent behavior.
- Complete feature set.
- Thin.
- Performance issues confined to base class.
- Portable, standard, open.

Comparison to Production OSI

- Problem representation
  - Prod: Representation "close to solver".
  - Devel: Common representation.
- Memory management
  - Prod: Cache managed in solver-specific layer (OsiXxxSolverInterface).
  - Devel: Cache managed in base-class layer (OsiSolverInterface).

Consistent Problem Representation/Behavior

- "Solver agnostic" vs. "solver independent"
  - 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.
- "Copy out" vs. "pointer out" invisible to user.

Solver Memory Management

- "Pointer out" (OSL, CLP)
  - Solver's representation of problem may be in solver-owned or user-owned (but restricted) memory.
  - Solver provides pointer to its representation to user.
  - Memory-efficient but dangerous in Fortran or C.
- "Copy out" (CPLEX, XPRESS)
  - Solver keeps closed copy of problem in memory it owns.
  - User provides memory to copy out vectors on query.
  - Safe but user must manage memory for his copy of problem data.

To provide efficient support for both solver styles, OSI must manage it's copy of problem data. User sees a "pointer out" style, but has some control.

Complete 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:
  - Hot starts
  - Pivot-level control
  - Equation solving with basis matrix (FTRAN/BTRAN)

Thin/Performance Issues Confined to Base Class

Goal: Minimize performance penalty for extra layer.

- Individual calls should be efficient.
- Efficiently manage cached copy of problem data.
- Allow user control of cache.
- Centralized cache manager in base class (vs. current one for every solver) so cache performance issues are confined to a single part of code.
- "Pointer out" solvers should not incur cache overhead.

Cache Management

- OsiXxxSolverInterface
  - **Get:**
    Query OSI 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 OSI base class.
- OsiSolverInterface
  - **Get:**
    Return cached copy.
  - **Set:**
    Update or delete cached copy.

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 OsiXxxSolverInterface).

Conclusion

Build a better solver interface. . .

Conclusion

Build a better solver interface. . .

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

Conclusion

Build a better solver interface. . .

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

- COIN-OR Users Group meeting, Sunday 6:30–7:30, Ballroom A8, Convention Center.
- Standards Meeting, Monday 8:45–9:45, Plaza Room, Fairmont.
- Watch for announcements on the coin-announce mailing list (sign up at www.coin-or.org).